MapReduce Service

Descripción general del servicio

Edición 01

Fecha 2023-09-15

HUAWEI TECHNOLOGIES CO., LTD.

Marcas y permisos


y otras marcas registradas de Huawei pertenecen a Huawei Technologies Co., Ltd.
Todas las demás marcas registradas y los otros nombres comerciales mencionados en este documento son propiedad de sus respectivos titulares.

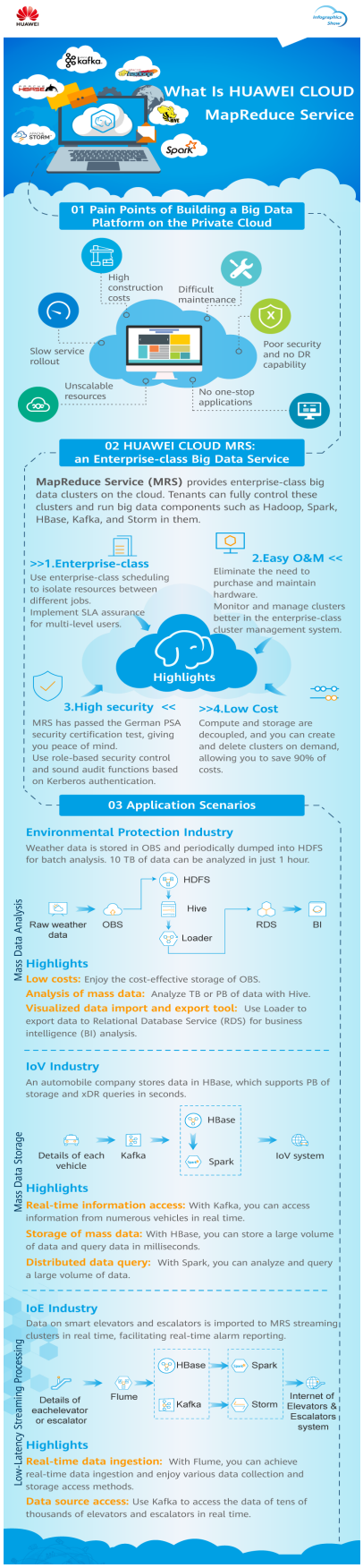
Aviso

Las funciones, los productos y los servicios adquiridos están estipulados en el contrato celebrado entre Huawei y el cliente. Es posible que la totalidad o parte de los productos, las funciones y los servicios descritos en el presente documento no se encuentren dentro del alcance de compra o de uso. A menos que el contrato especifique lo contrario, ninguna de las afirmaciones, informaciones ni recomendaciones contenidas en este documento constituye garantía alguna, ni expresa ni implícita.

La información contenida en este documento se encuentra sujeta a cambios sin previo aviso. En la preparación de este documento se realizaron todos los esfuerzos para garantizar la precisión de sus contenidos. Sin embargo, ninguna declaración, información ni recomendación contenida en el presente constituye garantía alguna, ni expresa ni implícita.

Índice

1 Infografías



2 ¿Qué es MRS?

La tecnología de big data se presenta como uno de los grandes desafíos a los que deberá hacer frente la era de Internet, ya que el volumen y los tipos de datos aumentan a niveles exponenciales. Las tecnologías de procesamiento de datos convencionales, como el almacenamiento en un solo nodo y las bases de datos relacionales, no pueden resolver los problemas emergentes inherentes a la tecnologías de big data. En este caso, Apache Software Foundation (ASF) ha lanzado una solución de código abierto para el procesamiento de big data Hadoop. Hadoop es una plataforma informática distribuida de código abierto que puede utilizar plenamente las capacidades informáticas y de almacenamiento de clústeres para procesar cantidades masivas de datos. Si las empresas despliega los sistemas de Hadoop por sí mismas, las desventajas incluyen altos costos, largo período de despliegue, mantenimiento difícil y uso inflexible.

Para solucionar estos problemas, el servicio MapReduce (MRS) se proporciona en Huawei Cloud para que pueda gestionar los componentes basados en Hadoop. Con MRS, puede desplegar un clúster de Hadoop con unos pocos clics. MRS proporciona clústeres de big data a nivel empresarial en la nube. Los inquilinos pueden controlar completamente los clústeres y ejecutar fácilmente componentes de big data como Storm, , Hadoop, Spark, HBase y Kafka. MRS es totalmente compatible con las API de código abierto e incorpora las ventajas de Huawei Cloud, la computación en la nube y el almacenamiento y la experiencia de la industria de big data para proporcionar a los clientes una plataforma de big data completa con alto rendimiento, bajo costo, flexibilidad y facilidad de uso. Además, la plataforma se puede personalizar en función de los requisitos de servicio para ayudar a las empresas a construir rápidamente un sistema de procesamiento de datos masivo y descubrir nuevos puntos de valor y oportunidades de negocio mediante el análisis y la extracción de cantidades masivas de datos en tiempo real o en tiempo no real.

Arquitectura del producto

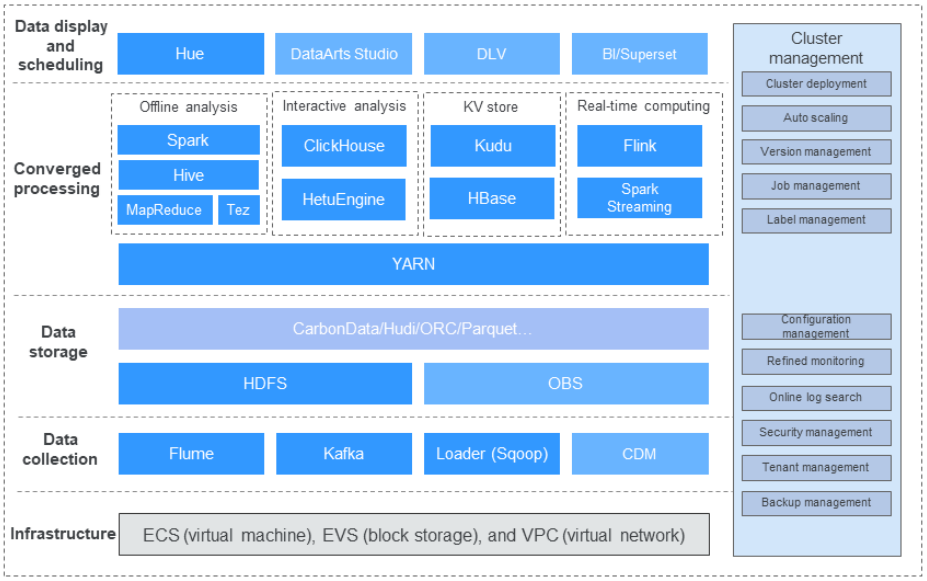
Lista de versiones de componentes de MRS enumera las versiones del componente de MRS.

Figura 2-1 muestra la arquitectura lógica de MRS.

NOTA

MRS 3.x o posterior no admite la gestión de parches en la consola de gestión.

Figura 2-1 Arquitectura de MRS



La arquitectura de MRS incluye infraestructura y fases de procesamiento de big data.

- Infraestructura

  Los clústeres de big data MRS se basan en Huawei Cloud Elastic Cloud Server (ECS), y utilizan plenamente las capacidades de alta confiabilidad y seguridad de la capa de virtualización.

  - Virtual Private Cloud (VPC) es una red interna virtual proporcionada para cada tenant. Está aislado de otras redes de forma predeterminada.
  - Elastic Volume Service (EVS) proporciona almacenamiento altamente confiable y de alto rendimiento.
  - ECS proporciona VM escalables y funciona con VPC, grupos de seguridad y el mecanismo de múltiples réplicas de EVS para crear un entorno informático eficiente, confiable y seguro.

- Recopilación de datos

  La capa de recopilación de datos proporciona la capacidad de importar datos de varias fuentes de dta, como Flume (ingestión de datos), Loader (importación de datos relacionales) y Kafka (cola de mensajes altamente confiable), a clústeres de big data MRS. Alternativamente, puede usar Cloud Data Migration (CDM) para importar datos externos a clústeres de MRS.

- Almacenamiento de datos

  Los clústeres de MRS pueden almacenar datos estructurados y no estructurados, y admiten múltiples formatos eficientes para satisfacer los requisitos de diferentes motores informáticos.

  - HDFS es un sistema de archivos distribuido de propósito general en una plataforma de big data.
  - OBS es un servicio de almacenamiento de objetos que ofrece alta disponibilidad y bajo costo.
  - HBase admite el almacenamiento de datos con índices y es aplicable a escenarios de consultas basadas en índices de alto rendimiento.

- Procesamiento de convergencia de datos
  - MRS proporciona múltiples motores de cómputo principales, incluidos MapReduce (procesamiento por lotes), Tez (modelo DAG), Spark (computación en memoria), Spark Streaming (computación de flujo por micro lotes), Storm (computación de flujo) y Flink (computación de flujo), para convertir datos estructuras y lógica en modelos de datos que cumplen con los requisitos de servicio en una variedad de escenarios de aplicaciones de big data.
  - Según el modelo de datos preestablecido y el análisis de datos de SQL fácil de usar, los usuarios pueden seleccionar Hive (almacenamiento de datos), SparkSQL y Presto (motor de consultas interactivas).
- Visualización y planificación de datos

  Muestra los resultados del análisis de datos y se integra con DataArts para proporcionar una plataforma de desarrollo colaborativo de big data única, lo que le ayuda a completar fácilmente múltiples tareas, como el modelado de datos, la integración de datos, el desarrollo de scripts, la programación de trabajos y la supervisión de O&M. hacer que el big data sea más accesible que nunca y ayudarle a construir sin esfuerzo centros de procesamiento de big data.
- Gestión de clústeres

  Todos los componentes del ecosistema de big data basado en Hadoop se despliega en modo distribuido, y su despliegue, gestión y operación son complejos.

  MRS proporciona una plataforma de gestión O&M unificada para la gestión de clústeres, que admite la implementación de clústeres con un solo clic, la selección de múltiples versiones, así como el escalado manual y el escalado automático de clústeres sin interrupción del servicio. Además, MRS proporciona gestión de trabajos, gestión de etiquetas de recursos y O&M de los componentes de procesamiento de datos anteriores en cada capa. También proporciona capacidades de operación de una sola parada, que abarcan monitoreo, informes de alarmas, configuración y actualización de parches.

Ventajas del producto

MRS tiene un potente equipo de núcleo Hadoop y se implementa basado en la plataforma de big data FusionInsight de Huawei. MRS se ha desplegado en decenas de miles de nodos y puede garantizar Acuerdos de nivel de servicio (SLA) para usuarios de varios niveles.

MRS tiene las siguientes ventajas:

- Alto rendimiento

  MRS admite la tecnología de almacenamiento de CarbonData desarrollada por sí misma. CarbonData es una solución de almacenamiento de big data de alto rendimiento. Permite que un conjunto de datos se aplique a múltiples escenarios y admite funciones, como la indexación de varios niveles, la codificación de diccionarios, la agregación previa, la partición dinámica y la consulta de datos en tiempo casi real. Esto mejora el escaneo de E/S y el rendimiento informático y devuelve los resultados del análisis de decenas de miles de millones de registros de datos en segundos. Además, MRS admite el planificador mejorado Superior de desarrollo propio, que rompe el cuello de botella de escala de un solo clúster y es capaz de planificar sobre nodos de 10,000 en un clúster.
- Rentabilidad

  Basado en una infraestructura en la nube diversificada, MRS ofrece varias opciones de computación y almacenamiento y separa la computación del almacenamiento, ofreciendo soluciones de almacenamiento masivo de datos rentables. MRS admite el escalado automático para abordar las cargas de servicio pico y fuera de pico, liberando recursos

inactivos en la plataforma de big data para los clientes. Los clústeres MRS se pueden crear y escalar cuando los necesite, y se pueden terminar o escalar después de usarlos, minimizando el costo.

- Alto nivel de seguridad

  MRS ofrece gestión de permisos de múltiples inquilinos de big data a nivel empresarial y gestión de seguridad para admitir el control de acceso y encriptación de datos basados en tablas y columnas.

- O&M Fácil

  MRS proporciona una plataforma de gestión de clústeres de big data visualizada, mejorando la eficiencia de operación. MRS admite la actualización continua de parches y proporciona información de liberación de parches visualizada y la instalación de parches con un solo clic sin intervención manual, lo que garantiza la estabilidad a largo plazo de los clústeres de usuarios.

- Alta confiabilidad

  La confiabilidad probada a gran escala y la estabilidad a largo plazo de MRS cumplen con los requisitos de alta confiabilidad de nivel empresarial. Además, MRS admite copias de respaldo automáticas de datos en zonas de disponibilidad y regiones, así como antiafinidad automática. Permite que las máquinas virtuales se distribuyan en diferentes máquinas físicas.

Usar MRS por primera vez

Si es la primera vez que lo usa, familiarícese con la siguiente información:

- **Conceptos básicos**

  Consulte **Componentes** y **Funciones** para aprender los conocimientos básicos de MRS, incluidos los principios básicos y las funciones mejoradas de cada componente de MRS, así como los conceptos y funciones únicos de MRS.

- **Pasos iniciales**

  Para saber cómo usar MRS, consulte **Pasos iniciales de MapReduce Service**. "Pasos iniciales" proporciona una guía de operación detallada de las muestras. Puede crear y utilizar clústeres de MRS según las instrucciones de operación.

- **Otras funciones y guías de operación**

  Si es un usuario de clúster MRS e ingeniero de O&M, puede realizar operaciones como la gestión del ciclo de vida del clúster, el escalado y la gestión de trabajos consultando **Guía de usuario de MapReduce Service**. Consulte **Guía de operación de componentes de MapReduce Service** para obtener información sobre cómo usar componentes en un clúster.

  Si es desarrollador, puede consultar la guía de operaciones y los proyectos de muestra en **Guía de desarrollo de MapReduce Service** de MRS para desarrollar, ejecutar y poner en marcha sus propias aplicaciones. También puede invocar a las API para gestionar clústeres MRS y ejecutar trabajos. Para obtener más información, consulte **Referencia de API de MapReduce Service**.

3 Ventajas de MRS en comparación con Hadoop de desarrollo propio

MRS proporciona clústeres de big data a nivel empresarial en la nube. Los inquilinos pueden controlar completamente los clústeres y ejecutar componentes de big data como Hadoop, Spark, HBase, Kafka y Storm con facilidad. MRS le libera de la compra y mantenimiento de hardware. MRS se basa en la plataforma de clase empresarial FusionInsight de Big Data de Huawei, y se ha implementado en decenas de miles de nodos en la industria, proporcionando garantía de SLA de varios niveles con soporte profesional del servicio del núcleo Hadoop. En comparación con los clústeres Hadoop de desarrollo propio, MRS presenta las siguientes ventajas:

1. **MRS admite la creación, eliminación y escalado de clústeres con un solo clic.** Para acceder a MRS Manager, utilice una dirección IP elástica (EIP), lo que facilitará el uso de clústeres de big data.
   - Los clústeres de big data autoconstruidos plantean problemas tales como altos costos, largos períodos, operaciones y mantenimiento difíciles e inflexibles. Para resolver estos problemas, MRS proporciona creación, eliminación, escalado horizontal y escalado de clúster con un solo clic, lo que le permite personalizar el tipo de clúster, el rango de componentes, el número de nodos de cada tipo, las especificaciones de VM, zonas de disponibilidad (AZs), red de VPC e información de autenticación. MRS puede crear automáticamente un clúster que cumpla con los requisitos de configuración. Además, puede crear rápidamente clústeres de múltiples aplicaciones, por ejemplo, clúster de análisis de Hadoop, clúster de HBase y clúster de Kafka. MRS admite el despliegue de clústeres heterogéneos. Es decir, las máquinas virtuales de diferentes especificaciones se pueden combinar en un clúster basado en los tipos de CPU, las capacidades de disco, los tipos de disco y los tamaños de memoria.
   - MRS proporciona un canal seguro basado en EIP para que pueda acceder fácilmente a las interfaces de usuario web de los componentes. Esto es más conveniente que vincular una EIP por sí mismo, y puede acceder a las interfaces de usuario web con unos pocos clics, evitando los pasos para iniciar sesión en una VPC, agregando reglas de grupo de seguridad y obteniendo una dirección IP pública.
   - MRS proporciona acciones de arranque personalizadas para configurar de forma flexible sus clústeres dedicados. El software de terceros que no es compatible con MRS se puede instalar automáticamente, lo que le permite realizar operaciones personalizadas, como modificar el entorno de ejecución del clúster.

- MRS admite la función WrapperFS, proporciona la capacidad de traducción de OBS (es decir, acceso a OBS a través de asignación de direcciones) y puede migrar datos sin problemas de HDFS a OBS. Después de la migración, puede acceder a los datos almacenados en OBS desde los clientes sin modificar la lógica del código de servicio.

2. **MRS admite el escalado automático, que es más rentable que el clúster de Hadoop de creación propia.**

   MRS admite el escalado automático para abordar las cargas de servicio pico y fuera de pico. Se aplica a recursos adicionales durante las horas pico y libera recursos inactivos durante las horas no pico, lo que le ayuda a ahorrar recursos inactivos en la plataforma de big data durante las horas no pico, minimizar los costos y centrarse en los servicios principales.

   En las aplicaciones de big data, especialmente en el análisis y procesamiento de datos periódicos, los recursos informáticos de clúster deben ajustarse dinámicamente en función de los cambios en los datos de servicio para cumplir con los requisitos de servicio. La función de escalado automático de MRS permite que los clústeres sean escalados elásticamente o en función de las cargas de clúster. Además, si el volumen de datos cambia regularmente y desea escalar o en un clúster antes de que cambie el volumen de datos, puede utilizar la función de plan de recursos de MRS. MRS admite dos tipos de políticas de escalado automático: reglas de escalado automático y planes de recursos

   - Reglas de escalado automático: puede aumentar o disminuir los nodos de tarea en función de las cargas de clúster en tiempo real. El escalado automático se activará cuando cambie el volumen de datos, pero puede haber algún retraso.
   - Planes de recursos: si el volumen de datos cambia periódicamente, puede crear planes de recursos para cambiar el tamaño del clúster antes de que cambie el volumen de datos, evitando así un retraso en el aumento o la disminución de recursos.

   Tanto las reglas de escalado automático como los planes de recursos pueden desencadenar escalado automático. Puede configurar ambos o configurar uno de ellos. La configuración de planes de recursos y reglas de escalado automático mejora la escalabilidad del nodo del clúster para hacer frente a picos de volumen de datos ocasionalmente inesperados.

3. **MRS admite el desacoplamiento de almacenamiento y cómputo, lo que mejora en gran medida la utilización de recursos de los clústeres de big data.**

   En la arquitectura tradicional de big data en la que se integran recursos de almacenamiento y cómputo, la ampliación es difícil y los recursos no se utilizan bien. Para resolver estos problemas, MRS adopta una arquitectura de separación de almacenamiento de información. Basado en OBS, el almacenamiento alcanza un 99.999999999% de confiabilidad y capacidad ilimitada, lo que soporta el crecimiento continuo de los datos empresariales. Los recursos informáticos se pueden escalar elásticamente de 0 a nodos *N*. Cientos de nodos se pueden aprovisionar rápidamente. Con la nueva arquitectura, los nodos informáticos pueden escalarse elásticamente. El almacenamiento de datos entre zonas de disponibilidad basado en OBS garantiza una mayor confiabilidad, le libera de preocuparse por emergencias como terremotos y cortes de fibra. Los recursos de almacenamiento y computación se pueden configurar de manera flexible y escalar elásticamente según sea necesario. Esto hace que la asignación de recursos sea más precisa y razonable, mejorando en gran medida la utilización de recursos de los clústeres de big data y reduciendo el costo de análisis integral en un 50%.

   Además, la arquitectura de separación de computación-almacenamiento de alto rendimiento rompe el límite de computación paralela de la arquitectura integrada de

computación-almacenamiento de información. Maximiza el alto ancho de banda y la alta concurrencia de OBS, y optimiza la eficiencia del acceso a los datos y la computación en paralelo en profundidad (como la operación de metadatos y la optimización del algoritmo de escritura) para mejorar un mayor rendimiento.

4. **MRS es compatible con CarbonData y Superior Scheduler de desarrollo propio, lo que brinda un mejor rendimiento.**
   - MRS admite la tecnología de almacenamiento de CarbonData desarrollada por sí misma. CarbonData es una solución de almacenamiento de big data de alto rendimiento. Permite que un conjunto de datos se aplique a múltiples escenarios y admite funciones, como la indexación de varios niveles, la codificación de diccionarios, la agregación previa, la partición dinámica y la consulta de datos en tiempo casi real. Esto mejora el escaneo de E/S y el rendimiento informático y devuelve los resultados del análisis de decenas de miles de millones de registros de datos en segundos.
   - Además, MRS admite el programador superior de desarrollo propio, que mejora la capacidad de escalado de un solo clúster y es capaz de programar más de 10,000 nodos en un clúster. Superior Scheduler es un motor de programación diseñado para el sistema de gestión de recursos distribuidos de Hadoop YARN. Es un programador de alto rendimiento y de nivel empresarial diseñado para grupos de recursos convergentes y requisitos de servicio de inquilinos múltiples. Superior Scheduler logra todas las funciones de programadores de código abierto, Fair Scheduler, y Capacity Scheduler. En comparación con los planificadores de código abierto, Superior Scheduler se ha mejorado en la política de planificación de recursos de múltiples inquilinos empresariales, el aislamiento de recursos y el uso compartido de múltiples usuarios en un inquilino, el rendimiento de planificación, la utilización de recursos del sistema y la escalabilidad del clúster, y está diseñado para reemplazar a los planificadores de código abierto.
5. **MRS optimiza el software y el hardware en función de los procesadores Kunpeng para liberar completamente la potencia informática del hardware y lograr la rentabilidad.**

   MRS soporta servidores de Kunpeng autodesarrollados cuyas capacidades multi-núcleo y de alta simultaneidad se utilizan completamente para proporcionar chips auto-optimizados de pila completa, y utiliza EulerOS autodesarrollados, Huawei JDK, y la capa de aceleración de datos para garantizar el rendimiento del hardware, entregando un alto poder de computación para la computación de big data. Con el rendimiento similar, el costo de la solución de big data de extremo a extremo se reduce en un 30%.
6. **MRS admite múltiples modos de aislamiento y gestión de permisos de múltiples inquilinos de big data de nivel empresarial, lo que garantiza una mayor seguridad.**
   - MRS admite la implementación de recursos y el aislamiento de recursos físicos en zonas dedicadas. Puede combinar de forma flexible recursos informáticos y de almacenamiento, como recursos informáticos dedicados + recursos de almacenamiento compartidos, recursos informáticos compartidos + recursos de almacenamiento dedicados y recursos informáticos dedicados + recursos de almacenamiento dedicados. Un clúster de MRS admite múltiples inquilinos lógicos. El aislamiento de permisos permite dividir los recursos informáticos, de almacenamiento y de tablas del clúster en función de los inquilinos.
   - Con la autenticación Kerberos, MRS proporciona funciones de control de acceso basado en roles (RBAC) y auditoría de sonido.
   - Con Cloud Trace Service (CTS) interconectado con MRS, se le proporcionan registros de operaciones de solicitudes de operación de recursos MRS y resultados

de solicitudes para consultas, auditorías y seguimiento. Puede usar CTS para auditar y rastrear todas las operaciones del clúster.

- Se ha demostrado que con Host Security Service (HSS) interconectado con MRS, la seguridad del servicio se mejora sin deteriorar las funciones y el rendimiento.
- MRS admite el inicio de sesión de usuario unificado basado en la interfaz de usuario web. El administrador proporciona autenticación de usuario, que le otorga permiso para acceder a un clúster.
- MRS admite el cifrado de almacenamiento de datos, el almacenamiento cifrado de todas las cuentas de usuario y contraseñas, la transmisión cifrada de canales de datos y la autenticación de certificado bidireccional para el acceso a datos de zonas de confianza cruzada de los módulos de servicio.
- Los clústeres de big data de MRS proporcionan una solución completa de múltiples inquilinos para big data de nivel empresarial. Multiinquilino hace referencia a una colección de varios recursos (cada conjunto de recursos es un inquilino) en un clúster de big data de MRS. Puede asignar y programar recursos, incluidos los recursos informáticos y de almacenamiento. Multitenant aísla los recursos de un clúster de big data en conjuntos de recursos. Los usuarios pueden arrendar los conjuntos de recursos deseados para ejecutar aplicaciones y trabajos y almacenar datos. En un clúster de big data, se pueden implementar varios conjuntos de recursos para satisfacer diversos requisitos de varios usuarios.
- MRS admite la gestión de permisos de grano fino. Con la capacidad de autorización detallada proporcionada por Huawei Cloud IAM, MRS puede especificar las operaciones, los recursos y las condiciones de solicitud de servicios específicos. Este mecanismo permite una autorización más flexible basada en políticas, cumpliendo los requisitos para un control de acceso seguro. Por ejemplo, puede conceder a los usuarios de MRS únicamente los permisos para realizar operaciones especificadas en clústeres de MRS, como crear un clúster y consultar una lista de clústeres en lugar de eliminar un clúster. Además, MRS admite la gestión de permisos de OBS para múltiples inquilinos. Los permisos para acceder a los depósitos y objetos de OBS en los depósitos se diferencian en función de los roles de usuario, de modo que los usuarios de MRS pueden controlar cada uno un directorio diferente en los buckets de OBS.
- MRS apoya la gestión de proyectos empresariales. El proyecto empresarial es una forma de gestionar los recursos en la nube. Enterprise Management proporciona servicios de gestión integrales para clientes empresariales, como recursos en la nube, personal, permisos y estados financieros. Las consolas de gestión comunes están orientadas al control y configuración de productos en la nube individual. Por el contrario, la consola de Enterprise Management está más centrada en la gestión de recursos. Está diseñado para ayudar a las empresas a gestionar recursos, personal, permisos y finanzas basados en la nube, de manera jerárquica, como la gestión de empresas, departamentos y proyectos. MRS permite a los usuarios que han habilitado Enterprise Project Management Service (EPS) configurar proyectos empresariales para un clúster durante la creación de clústeres y utilizar EPS para gestionar recursos MRS por grupo. Esta función es aplicable a escenarios en los que necesita gestionar varios recursos por grupo y realizar operaciones como el control de permisos y la consulta de tarifas basada en proyectos de empresa.

7. **MRS implementa HA para todos los nodos de gestión y admite un mecanismo de confiabilidad integral, lo que hace que el sistema sea más confiable.**

   Basado en el software de código abierto de Apache Hadoop, MRS optimiza y mejora la fiabilidad de los principales componentes del servicio.

- HA para todos los nodos de gestión

  En la versión de código abierto de Hadoop, los datos y los nodos de cómputo se gestionan en un sistema distribuido, en el que un único punto de fallo (SPOF) no afecta al funcionamiento de todo el sistema. Sin embargo, un SPOF puede ocurrir en nodos de gestión que se ejecutan en modo centralizado, lo que se convierte en la debilidad de la confiabilidad global del sistema.

  MRS proporciona mecanismos de doble nodo similares para todos los nodos de gestión de los componentes de servicio, como Manager, Presto, HDFS NameNodes, Hive Servers, HBase HMasters, YARN Resource Managers, Kerberos Servers, y Ldap Servers. Todos ellos se implementan en modo activo/en espera o se configuran con carga compartida, lo que evita que los SPOF afecten la confiabilidad del sistema.

- Mecanismo de confiabilidad integral

  Mediante el análisis de confiabilidad, se proporcionan las siguientes medidas para manejar excepciones de software y hardware para mejorar la confiabilidad del sistema:

  - Después de restaurar la fuente de alimentación, los servicios se ejecutan correctamente independientemente de un fallo de alimentación de un solo nodo o de todo el clúster, lo que garantiza la confiabilidad de los datos en caso de fallos de alimentación inesperados. Los datos clave no se perderán a menos que el disco duro esté dañado.
  - Las comprobaciones del estado de salud y el manejo de fallos del disco duro no afectan a los servicios.
  - Las fallas del sistema de archivos se pueden manejar automáticamente y los servicios afectados se pueden restaurar automáticamente.
  - Las fallas del proceso y del nodo se pueden manejar automáticamente, y los servicios afectados se pueden restaurar automáticamente.
  - Las fallas de la red se pueden manejar automáticamente y los servicios afectados se pueden restaurar automáticamente.

8. **MRS proporciona una interfaz de gestión de clústeres de big data visualizada de manera unificada, lo que facilita la operación y el mantenimiento.**

   - En la interfaz de gestión de clústeres de big data, están disponibles el inicio y la detención del servicio, la modificación de la configuración y la comprobación de estado. MRS también proporciona funciones visualizadas y convenientes de gestión, monitoreo y alarma de clústeres. Además, puede comprobar y auditar el estado del sistema con un solo clic, lo que garantiza el funcionamiento normal del sistema y reduce los costos de operación del sistema.
   - Después de que se configura la notificación de mensaje simple (SMN), MRS puede enviar información del estado de funcionamiento del clúster en tiempo real, incluidos los cambios del clúster y las alarmas de componentes en tiempo real a través de mensajes SMS o correos electrónicos, lo que facilita el monitoreo en tiempo real, y envío de alarmas en tiempo real.
   - MRS admite la actualización continua de parches y proporciona información de liberación de parches visualizada y la instalación de parches con un solo clic sin intervención manual, lo que garantiza la estabilidad a largo plazo de los clústeres de usuarios.
   - Si se produce un problema al utilizar un clúster MRS, puede iniciar la autorización O&M en la consola de gestión de MRS. El personal de O&M puede ayudarlo a localizar rápidamente el problema, y puede revocar la autorización en cualquier

momento. También puede iniciar el uso compartido de registros en la consola de gestión de MRS para compartir un ámbito de log especificado con el personal de O&M, de modo que el personal de O&M pueda localizar fallas sin tener acceso al clúster.

- MRS admite el volcado de registros sobre fallas de creación de clústeres en OBS para que el personal de O&M obtenga y analice logs.

9. **MRS tiene un ecosistema abierto y admite una interconexión perfecta con servicios periféricos, lo que le permite crear rápidamente una plataforma unificada de big data.**
   - Basado en MRS, un servicio de big data de pila completa, las empresas pueden construir una plataforma de big data unificada para la ingesta de datos, almacenamiento, análisis y minería de valor con un solo clic, e interconectarse con el estudio de DataArts y los servicios de visualización de datos para ayudar a los clientes a migrar fácilmente los datos a la nube. desarrollar y programar trabajos de big data y mostrar datos. Esto libera a los clientes de la construcción de plataformas de big data complejas y de la calibración y el mantenimiento profesionales de big data para que los clientes puedan centrarse más en las aplicaciones de la industria y utilizar una copia de datos en múltiples escenarios de servicio. DataArts es una plataforma integral de operaciones de desarrollo del ciclo de vida de los datos que ofrece una amplia gama de funciones, como integración de datos, desarrollo, gobernanza, servicio y visualización. Los datos de MRS se pueden ingerir en DataArts Studio para el desarrollo colaborativo visualizado con un solo clic aprovechando la interfaz gráfica de usuario visualizada de DataArts Studio, abundantes tipos de desarrollo de datos (guión y trabajo), programación de trabajos totalmente hospedada y monitoreo de O&M, y canalizaciones de procesamiento de datos de la industria integradas. Esto hace que el big data sea mucho más fácil de usar, le ayuda a construir rápidamente centros de procesamiento de big data y permite una monetización rápida.
   - MRS es totalmente compatible con el ecosistema de big data de código abierto. Con abundantes herramientas de migración de datos y aplicaciones, MRS le ayuda a migrar datos rápidamente desde sus propias plataformas sin modificación de código ni interrupción del servicio.

4 Escenarios de aplicación

Big data es omnipresente en nuestras vidas. Huawei Cloud MRS es adecuado para procesar big data en sectores como el Internet de las cosas (IoT), el comercio electrónico, las finanzas, la fabricación, la salud, la energía y los departamentos gubernamentales.

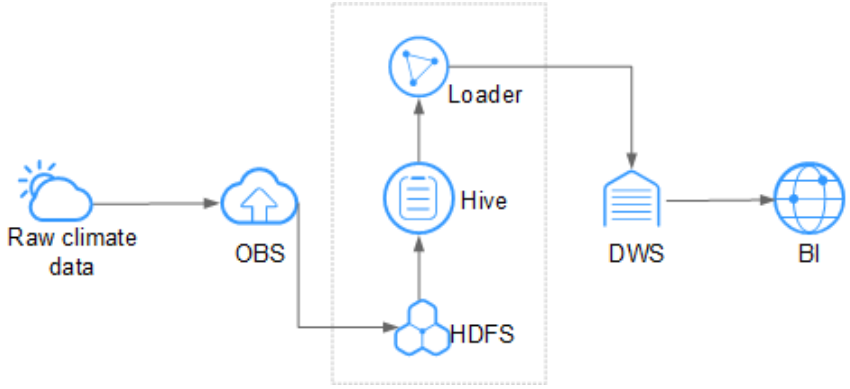
Análisis de datos a gran escala

El análisis de datos a gran escala es un escenario importante en los sistemas modernos de big data. En general, una empresa tiene múltiples fuentes de datos. Después de acceder a los datos, se requiere el procesamiento de extracción, transformación y carga (ETL) para generar datos modelizados para cada módulo de servicio para analizar y clasificar los datos. Este tipo de servicio tiene las siguientes características:

- Los requisitos para la ejecución en tiempo real no son altos, y el tiempo de ejecución del trabajo varía de docenas de minutos a horas.
- El volumen de datos es grande.
- Hay varias fuentes de datos y formatos diversificados.
- El procesamiento de datos generalmente consiste en múltiples tareas, y los recursos deben planificarse en detalle.

En la industria de la protección del medio ambiente, los datos climáticos se almacenan en OBS y periódicamente se vierten en HDFS para el análisis por lotes. 10 TB de datos climáticos se pueden analizar en 1 hora.

Figura 4-1 Análisis de datos a gran escala en la industria de la protección del medio ambiente



MRS tiene las siguientes ventajas en este escenario.

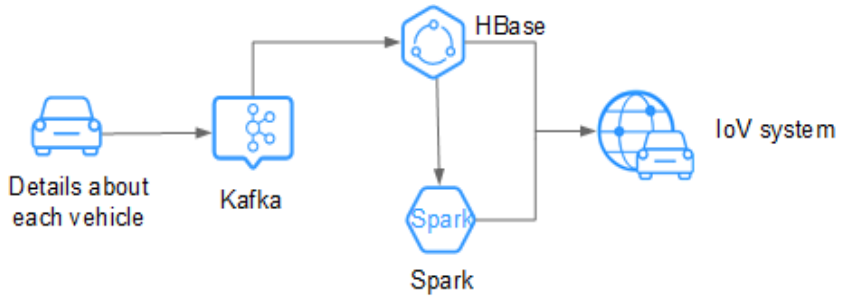
- Bajo costo: OBS ofrece almacenamiento rentable.
- Análisis masivo de datos: los datos a nivel de TB/PB se analizan mediante Hive.
- Herramienta de importación y exportación de datos visualizados: Loader exporta datos a Data Warehouse Service (DWS) para análisis de inteligencia empresarial (BI).

Almacenamiento de datos a gran escala

Un usuario que tiene una gran cantidad de datos estructurados generalmente requiere capacidades de consulta en tiempo casi real basadas en índices. Por ejemplo, en un escenario de Internet de Vehículos (IoV), la información de mantenimiento del vehículo es consultada por el número del vehículo. Por lo tanto, la información del vehículo se indexa basándose en los números del vehículo cuando se está almacenando, para implementar una respuesta de segundo nivel en este escenario. Generalmente, el volumen de datos es grande. El usuario puede almacenar datos durante uno a tres años.

Por ejemplo, en la industria de IoV, una empresa de automóviles almacena datos en HBase, que admite el almacenamiento a nivel de PB y las consultas CDR en milisegundos.

Figura 4-2 Almacenamiento de datos a gran escala en la industria IoV



MRS tiene las siguientes ventajas en este escenario.

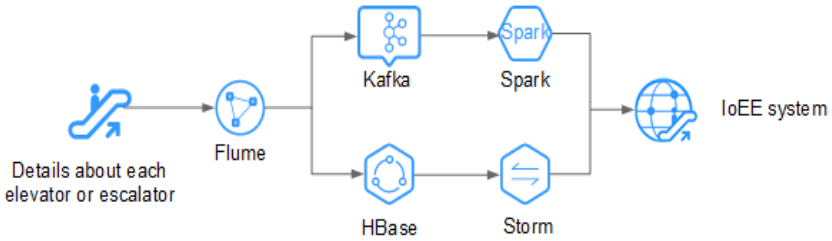
- Tiempo real: Kafka accede a grandes cantidades de mensajes del vehículo en tiempo real.
- Almacenamiento masivo de datos: HBase almacena volúmenes masivos de datos y admite consultas de datos en milisegundos.
- Consulta de datos distribuidos: Spark analiza y consulta volúmenes masivos de datos.

Procesamiento de datos en tiempo real

El procesamiento de datos en tiempo real se utiliza generalmente en escenarios como detección de anomalías, detección de fraudes, alarmas basadas en reglas y monitoreo de procesos de servicio. Los datos se procesan mientras se introducen en el sistema.

Por ejemplo, en la industria de Internet of elevators & escalators (IoEE), los datos de ascensores y escaleras mecánicas inteligentes se importan a los clústeres de streaming de MRS en tiempo real para alarmar en tiempo real.

Figura 4-3 Procesamiento de streaming de baja latencia en la industria IoEE



MRS tiene las siguientes ventajas en este escenario.

- Ingestión de datos en tiempo real: Flume implementa la ingestión de datos en tiempo real y proporciona varios métodos de recopilación de datos y acceso al almacenamiento.
- Acceso a la fuente de datos: Kafka accede a los datos de decenas de miles de ascensores y escaleras mecánicas en tiempo real.

5 Elección de una versión apropiada al comprar un clúster de MRS

Versiones del clúster de MRS

- **Normal**
  - **Funciones**

    Esta versión normal proporciona operaciones básicas de clúster, como configuración, gestión y O&M. Para obtener más información, consulte **Guía del usuario del MapReduce Service**.
  - **Componentes**

    Esta versión normal admite componentes como Presto, Impala, Kudu, Sqoop y IoTDB. Puede seleccionar componentes de diferentes versiones en esta versión. Para obtener más información, consulte **Lista de versiones de componentes de MRS** y **Guía de funcionamiento del componente de MapReduce Service**.
- **LTS (long term support)**
  - **Funciones**

    Además de las operaciones básicas del clúster, la versión LTS admite la actualización de la versión.

    La versión LTS admite el despliegue de multi-AZ.
  - **Componentes**

    La versión LTS soporta el componente de HetuEngine además de los componentes soportados por la versión normal. Para obtener más información, consulte **Guía de operación del componente de MapReduce Service**.

Sugerencias de compra

- Los clústeres de la versión LTS admiten la actualización de la versión. Para que sus clústeres sean actualizables, elija la versión LTS.
- Los clústeres de la versión LTS se pueden implementar en diferentes zonas de disponibilidad para implementar DR entre zonas de disponibilidad. Para que sus clústeres sean más seguros y tengan mayores capacidades de recuperación ante desastres, elija la versión LTS.
- Los clústeres de la versión LTS admiten el componente de HetuEngine. Para usar HetuEngine, escoja la versión LTS.

NOTA

Después de la compra, la versión LTS no se puede cambiar a la versión normal. Elija una versión adecuada según las necesidades de su negocio.

Diferencias de facturación entre versiones de clúster MRS

Las versiones normal y LTS tienen diferentes funciones y, por lo tanto, se facturan de manera diferente. Para obtener más información, consulte **Facturación**. También puede utilizar calculadora de precio para calcular rápidamente el precio de referencia de un clúster MRS seleccionando la versión del clúster y las especificaciones de nodo que necesita.

6 Componentes

6.1 Lista de versiones de componentes de MRS

Componentes y versiones

Tabla 6-1 enumera los componentes y sus versiones requeridas por cada versión del clúster de MRS.

NOTA

- Hadoop incluye HDFS, YARN y MapReduce.
- DBService, ZooKeeper, KrbServer y LdapServer son componentes utilizados en el clúster y no se muestran durante la creación del clúster.
- Las versiones de componentes de MRS deben ser coherentes con las versiones de componentes de código abierto.

Tabla 6-1 Versiones de componentes de MRS

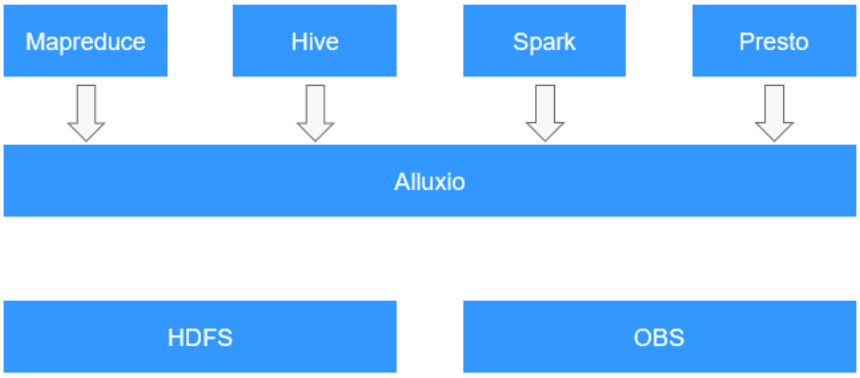
Componente	MRS 1.9.2 (Aplicable a MRS 1.9.*x*)	MRS 3.1.0	MRS 3.1.2-LTS	MRS 3.1.5
Alluxio	2.0.1	-	-	-
CarbonData	1.6.1	2.0.1	2.2.0	2.2.0
ClickHouse	-	21.3.4.25	21.3.4.25	21.3.4.25
DBService	1.0.0	2.7.0	2.7.0	2.7.0
Flink	1.7.0	1.12.0	1.12.2	1.12.2
Flume	1.6.0	1.9.0	1.9.0	1.9.0
HBase	1.3.1	2.2.3	2.2.3	2.2.3
HDFS	2.8.3	3.1.1	3.1.1	3.1.1
HetuEngine	-	-	1.2.0	-

Componente	MRS 1.9.2 (Aplicable a MRS 1.9.*x*)	MRS 3.1.0	MRS 3.1.2-LTS	MRS 3.1.5
Hive	2.3.3	3.1.0	3.1.0	3.1.0
Hudi	-	0.7.0	0.9.0	0.9.0
Hue	3.11.0	4.7.0	4.7.0	4.7.0
Impala	-	3.4.0	-	3.4.0
IoTDB	-	-	-	0.12.0
Kafka	1.1.0	2.11-2.4.0	2.11-2.4.0	2.11-2.4.0
KafkaManager	1.3.3.1	-	-	-
KrbServer	1.15.2	1.17	1.18	1.17
Kudu	-	1.12.1	-	1.12.1
LdapServer	1.0.0	2.7.0	2.7.0	2.7.0
Loader	2.0.0	-	1.99.3	-
MapReduce	2.8.3	3.1.1	3.1.1	3.1.1
Oozie	-	5.1.0	5.1.0	5.1.0
OpenTSDB	2.3.0	-	-	-
Presto	0.216	333	-	333
Phoenix (integrated in HBase)	-	5.0.0	5.0.0	5.0.0
Ranger	1.0.1	2.0.0	2.0.0	2.0.0
Spark	2.2.2	-	-	-
Spark2x	-	2.4.5	3.1.1	3.1.1
Sqoop	-	1.4.7	-	1.4.7
Storm	1.2.1	-	-	-
Tez	0.9.1	0.9.2	0.9.2	0.9.2
YARN	2.8.3	3.1.1	3.1.1	3.1.1
ZooKeeper	3.5.1	3.5.6	3.6.3	3.6.3
MRS Manager	1.9.2	-	-	-
FusionInsight Manager	-	8.1.0	8.1.2	8.1.0

6.2 Alluxio

Alluxio es tecnología de orquestación de datos para análisis e IA para la nube. En el ecosistema de big data de MRS, Alluxio se encuentra entre la computación y el almacenamiento. Proporciona una capa de abstracción de datos para los marcos de computación que incluyen Apache Spark, Presto, MapReduce y Apache Hive, para que las aplicaciones de computación de capa superior puedan acceder a los sistemas de almacenamiento persistentes que incluyen HDFS y OBS a través de las API de cliente unificadas y un espacio de nombres global. De esta manera, la informática y el almacenamiento están separados.

Figura 6-1 Arquitectura de Alluxio



Ventajas:

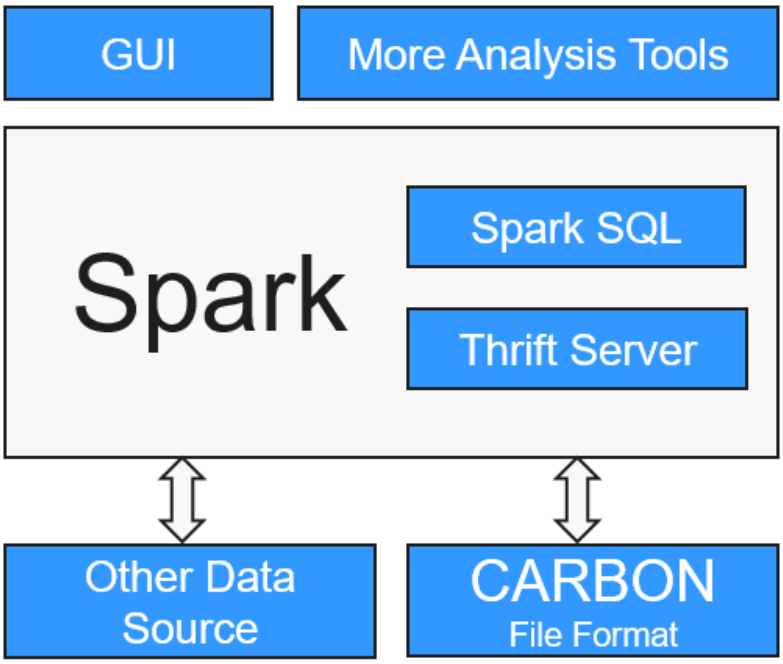
- Proporciona rendimiento de E/S en memoria y hace que las aplicaciones basadas en datos de escalamiento elástico sean rentables.
- Acceso simplificado al almacenamiento de objetos y a la nube
- Gestión de datos simplificada y un único punto de acceso a múltiples fuentes de datos
- Fácil implementación de aplicaciones

Para obtener más información sobre Alluxio, visite **https://docs.alluxio.io/os/user/stable/en/Overview.html**.

6.3 CarbonData

CarbonData es un nuevo formato de almacén de datos nativo de Apache Hadoop. CarbonData permite consultas interactivas más rápidas sobre PetaBytes de datos utilizando técnicas avanzadas de almacenamiento de columnas, índice, compresión y codificación para mejorar la eficiencia informática. Además, el CarbonData es también un motor de análisis de alto rendimiento que integra fuentes de datos con Spark.

Figura 6-2 Arquitectura básica de CarbonData



El propósito de utilizar CarbonData es proporcionar una respuesta rápida a consultas de ad hoc de big data. Esencialmente, CarbonData es un motor de procesamiento analítico en línea (OLAP), que almacena datos utilizando tablas similares a las del sistema de gestión de bases de datos relacionales (RDBMS). Puede importar más de 10 TB de datos a tablas creadas en formato CarbonData y CarbonData organiza y almacena automáticamente los datos mediante los índices multidimensionales comprimidos. Después de cargar los datos en CarbonData, CarbonData responde a consultas de ad hoc en segundos.

CarbonData integra fuentes de datos en el ecosistema de Spark. Puede utilizar Spark SQL para consultar y analizar datos, o utilizar la herramienta de terceros ThriftServer proporcionada por Spark para conectarse a Spark SQL.

Caracterísiticas de CarbonData

- SQL CarbonData es compatible con Spark SQL y admite operaciones de consulta SQL realizadas en Spark SQL.
- Definición de conjunto de datos de table simple: CarbonData le permite definir y crear conjuntos de datos mediante sentencias de lenguaje de definición de datos (DDL) fáciles de usar. CarbonData DDL es flexible y fácil de usar, y puede definir tables complejos.
- Fácil gestión de datos: CarbonData ofrece varias funciones de gestión de datos para la carga y el mantenimiento de datos. Puede cargar datos históricos y cargar datos nuevos de forma incremental. Los datos cargados se pueden eliminar de acuerdo con el tiempo de carga y las operaciones de carga de datos específicos se pueden cancelar.
- El formato de archivo CarbonData es un almacén de columnas en HDFS. Tiene muchas características que tiene un formato de columna moderno, como el esquema de compresión y divisible.

Características únicas de CarbonData

- Almacena datos junto con el índice: Acelera significativamente el rendimiento de la consulta y reduce los análisis de E/S y los recursos de CPU, cuando hay filtros en la

consulta. El índice de CarbonData consta de múltiples niveles de índices. Un marco de procesamiento puede aprovechar este índice para reducir la tarea que necesita programar y procesar, y también puede realizar omitir el escaneo en una unidad de grano más fino (llamada blocklet) en el escaneo del lado de la tarea en lugar de escanear todo el archivo.

- Datos codificados operables: A través de la compatibilidad con esquemas de compresión y codificación globales eficientes, CarbonData puede consultar datos comprimidos/codificados. Los datos pueden ser convertidos justo antes de devolver los resultados a los usuarios, que es "materializado tarde".
- Soporta varios casos de uso con un solo formato de datos: como consulta interactiva de estilo OLAP, acceso secuencial (big scan) y acceso aleatorio (narrow scan).

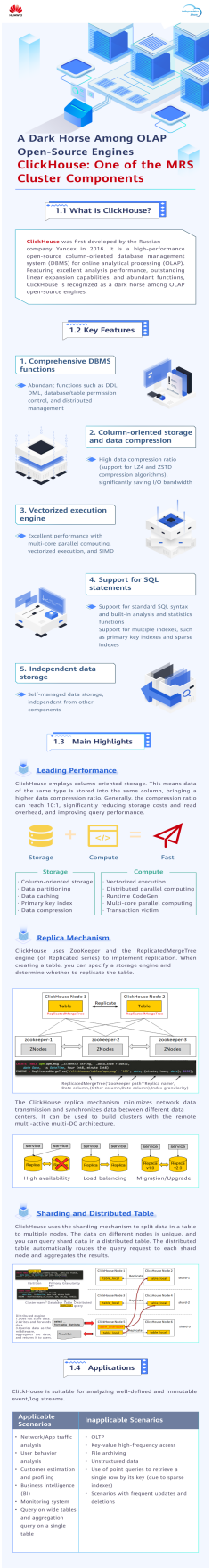
Tecnologías y ventajas clave de CarbonData

- Respuesta rápida a la consulta: CarbonData ofrece consultas de alto rendimiento. La velocidad de consulta de CarbonData es 10 veces mayor que la de Spark SQL. Utiliza formatos de datos dedicados y aplica múltiples tecnologías de índice, código de diccionario global y múltiples optimizaciones push-down, proporcionando una respuesta rápida a las consultas de datos a nivel de TB.
- Compresión de datos eficiente: CarbonData comprime los datos combinando los algoritmos de compresión ligeros y pesados. Esto ahorra significativamente entre un 60% y un 80% de espacio de almacenamiento de datos y el costo de almacenamiento de hardware.

Para obtener más información sobre la arquitectura y los principios de CarbonData, consulte **https://carbondata.apache.org/**.

6.4 ClickHouse

6.4.1 Infografías para ClickHouse


A Dark Horse Among OLAP Open-Source Engines
ClickHouse: One of the MRS Cluster Components
1.1 What Is ClickHouse?
ClickHouse was first developed by the Russian company Yandex in 2016. It is a high-performance open-source column-oriented database management system (DBMS) for online analytical processing (OLAP). Featuring excellent analysis performance, outstanding linear expansion capabilities, and abundant functions, ClickHouse is recognized as a dark horse among OLAP open-source engines.
1.2 Key Features
1. Comprehensive DBMS functions
Abundant functions such as DDL, DML, database/table permission control, and distributed management
2. Column-oriented storage and data compression
High data compression ratio (support for LZ4 and ZSTD compression algorithms), significantly saving I/O bandwidth
3. Vectorized execution engine
Excellent performance with multi-core parallel computing, vectorized execution, and SIMD
4. Support for SQL statements
Support for standard SQL syntax and built-in analysis and statistics functions
Support for multiple indexes, such as primary key indexes and sparse indexes
5. Independent data storage
Self-managed data storage, independent from other components
1.3 Main Highlights
Leading Performance
ClickHouse employs column-oriented storage. This means data of the same type is stored into the same column, bringing a higher data compression ratio. Generally, the compression ratio can reach 10:1, significantly reducing storage costs and read overhead, and improving query performance.
Storage
Compute
Fast
Storage
Column-oriented storage
Data partitioning
Data caching
Primary key index
Data compression
Compute
Vectorized execution
Distributed parallel computing
Runtime CodeGen
Multi-core parallel computing
Transaction victim
Replica Mechanism
ClickHouse uses ZooKeeper and the ReplicatedMergeTree engine (of Replicated series) to implement replication. When creating a table, you can specify a storage engine and determine whether to replicate the table.
ClickHouse Node 1
Table
Replicate
ClickHouse Node 2
Table
zookeeper-1
ZNodes
zookeeper-2
ZNodes
zookeeper-3
ZNodes
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.
High availability
Load balancing
Migration/Upgrade
Sharding and Distributed Table
ClickHouse uses the sharding mechanism to split data in a table to multiple nodes. The data on different nodes is unique, and you can query shard data in a distributed table. The distributed table automatically routes the query request to each shard node and aggregates the results.
1.4 Applications
ClickHouse is suitable for analyzing well-defined and immutable event/log streams.
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
Key-value high-frequency access
File archiving
Unstructured data
Use of point queries to retrieve a single row by its key (due to sparse indexes)
Scenarios with frequent updates and deletions

6.4.2 ClickHouse

Introducción a ClickHouse

ClickHouse es una base de datos columnar de código abierto orientada al análisis y procesamiento en línea. Es independiente del sistema de big data Hadoop y cuenta con la máxima tasa de compresión y un rendimiento de consulta rápida. Además, ClickHouse admite consultas SQL y proporciona un buen rendimiento de consultas, especialmente el análisis de agregación y el rendimiento de consultas basado en tablas grandes y amplias. La velocidad de consulta es un orden de magnitud más rápida que la de otras bases de datos analíticas.

Las funciones principales de ClickHouse son las siguientes:

Funciones de DBMS completas

ClickHouse es un sistema de gestión de bases de datos (DBMS) que proporciona las siguientes funciones básicas:

- Lenguaje de definición de datos (DDL): permite crear, modificar o eliminar dinámicamente bases de datos, tablas y vistas sin reiniciar los servicios.
- Lenguaje de manipulación de datos (DML): permite consultar, insertar, modificar o eliminar dinámicamente los datos.
- Control de permisos: admite la configuración de permisos de operación de tablas o bases de datos basadas en el usuario para garantizar la seguridad de los datos.
- Copia de respaldo y restauración de datos: admite copias de respaldo, exportación, importación y restauración de datos para cumplir con los requisitos del entorno de producción.
- Gestión distribuida: proporciona el modo de clúster para gestionar automáticamente múltiples nodos de base de datos.

Almacenamiento basado en columnas y compresión de datos

ClickHouse es una base de datos que utiliza almacenamiento basado en columnas. Los datos se organizan por columna. Los datos en la misma columna se almacenan juntos, y los datos en columnas diferentes se almacenan en archivos diferentes.

Durante la consulta de datos, el almacenamiento en columna puede reducir el rango de exploración de datos y el tamaño de transmisión de datos, mejorando así la eficiencia de la consulta de datos.

En un sistema de base de datos tradicional basado en filas, los datos se almacenan en la secuencia de **Tabla 6-2**:

Tabla 6-2 Base de datos basada en filas

row	ID	Flag	Name	Event	Time
0	12345678901	0	name1	1	2020/1/11 15:19
1	32345678901	1	name2	1	2020/5/12 18:10
2	42345678901	1	name3	1	2020/6/13 17:38

row	ID	Flag	Name	Event	Time
N	...	...	...	...	...

En una base de datos basada en filas, los datos de la misma fila se almacenan físicamente juntos. En un sistema de base de datos basado en columnas, los datos se almacenan en la secuencia de **Tabla 6-3**:

Tabla 6-3 Base de datos de columnas

row:	0	1	2	N
ID:	12345678901	32345678901	42345678901	...
Flag:	0	1	1	...
Name:	name1	name2	name3	...
Event:	1	1	1	...
Time:	2020/1/11 15:19	2020/5/12 18:10	2020/6/13 17:38	...

Este ejemplo muestra solo la disposición de los datos en una base de datos de columnas. Las bases de datos de columnas almacenan los datos en la misma columna juntas y los datos en diferentes columnas por separado. Las bases de datos de columnas son más adecuadas para escenarios de procesamiento analítico en línea (OLAP).

Ejecutor vectorizado

ClickHouse utiliza datos múltiples de instrucciones únicas (SIMD) de la CPU para implementar la ejecución vectorizada. SIMD es un modo de implementación que utiliza una única instrucción para operar múltiples piezas de datos y mejora el rendimiento con paralelismo de datos (otros métodos incluyen paralelismo a nivel de instrucción y paralelismo a nivel de subproceso). El principio de SIMD es implementar operaciones de datos en paralelo a nivel de registro de CPU.

Modelo relacional y consulta SQL

ClickHouse utiliza SQL como lenguaje de consulta y proporciona API de consulta SQL estándar para que los sistemas de visualización de análisis de terceros se integren fácilmente con ClickHouse.

Además, ClickHouse utiliza un modelo relacional. Por lo tanto, el costo de migrar el sistema construido en una base de datos relacional tradicional o almacén de datos a ClickHouse es menor.

Particionamiento de datos y consulta distribuida

El clúster ClickHouse consta de una o más particiones, y cada partición corresponde a un nodo de servicio de ClickHouse. El número máximo de particiones depende del número de nodos (una partición corresponde a un solo nodo de servicio).

ClickHouse presenta los conceptos de tabla local y tabla distribuida. Una tabla local es equivalente a una partición de datos. Una tabla distribuida en sí misma no almacena ningún

dato. Es un proxy de acceso de la tabla local y funciona como middleware de partición. Con la ayuda de tablas distribuidas, se puede acceder a múltiples particiones de datos usando el proxy, implementando así una consulta distribuida.

Aplicaciones de ClickHouse

ClickHouse es la abreviatura de Click Stream y Data Warehouse. Inicialmente se aplica a una herramienta de análisis de tráfico web para realizar análisis OLAP para almacenes de datos basados en flujos de eventos de clic en la página. Actualmente, ClickHouse es ampliamente utilizado en publicidad en Internet, análisis de tráfico de aplicaciones y web, telecomunicaciones, finanzas e Internet de las cosas (IoT). Es aplicable a escenarios de aplicaciones de inteligencia empresarial y tiene un gran número de aplicaciones y prácticas en todo el mundo. Para obtener más información, visite **https://clickhouse.tech/docs/en/introduction/adopters/**.

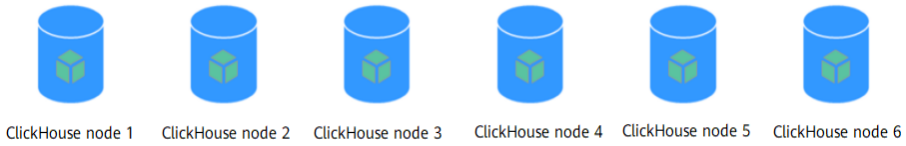
Funciones mejoradas de código abierto de ClickHouse

ClickHouse de MRS tiene ventajas como el modo de clúster automático, , despliegue de HA, y escalamiento suave y elástico.

- Modo de clúster automático

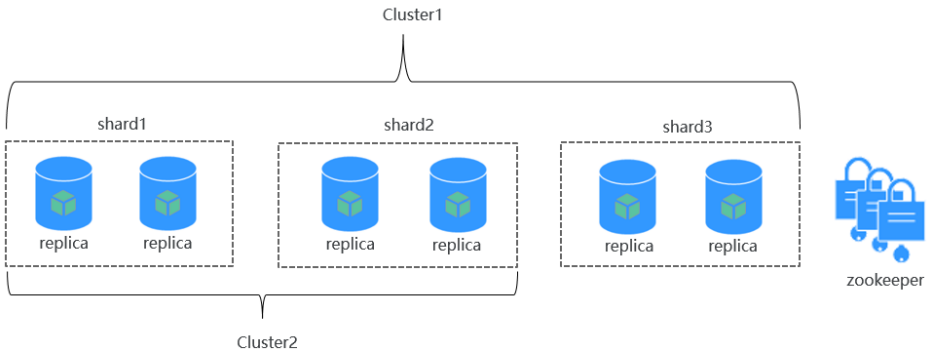
  Como se muestra en **Figura 6-3**, un clúster consta de múltiples nodos de ClickHouse que no tiene nodo central. Es más bien un fondo de recursos estático. Si se utiliza el modo de clúster de ClickHouse para los servicios, debe predefinir la información del clúster en el archivo de configuración de cada nodo. Solo de esta manera se puede acceder correctamente a los servicios.

  Figura 6-3 Clúster de ClickHouse

  Los usuarios desconocen las particiones de datos y el almacenamiento de réplicas en sistemas de bases de datos comunes. Sin embargo, ClickHouse le permite planificar y definir de forma proactiva configuraciones detalladas como fragmentos, particiones y ubicaciones de réplicas. La instancia de ClickHouse de MRS empaqueta el trabajo de manera unificada y lo adapta al modo automático, implementando una gestión unificada, que es flexible y fácil de usar. Una instancia de ClickHouse consta de tres nodos de ZooKeeper y varios nodos de ClickHouse. El modo de réplica dedicada se utiliza para garantizar una alta confiabilidad de las copias de datos duales.

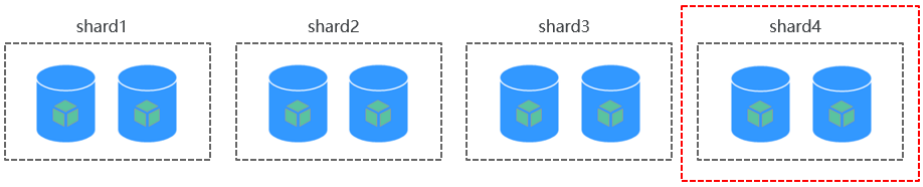
Figura 6-4 Estructura del clúster de ClickHouse



- Escala lisa y elástica

  A medida que el negocio crece rápidamente, MRS proporciona ClickHouse, una herramienta de migración de datos, para escenarios como la capacidad de almacenamiento del clúster o los recursos de cómputo de la CPU que se acercan al límite. Esta herramienta se utiliza para migrar algunas particiones de una o varias tablas de MergeTree en varios nodos de ClickHouseServer a las mismas tablas en otros nodos de ClickHouseServer. De esta manera, se garantiza la disponibilidad del servicio y se implementa una expansión de la capacidad sin problemas.

  Cuando agregue nodos de ClickHouse a un clúster, utilice esta herramienta para migrar algunos datos de los nodos existentes a los nuevos para equilibrar los datos después de la expansión.

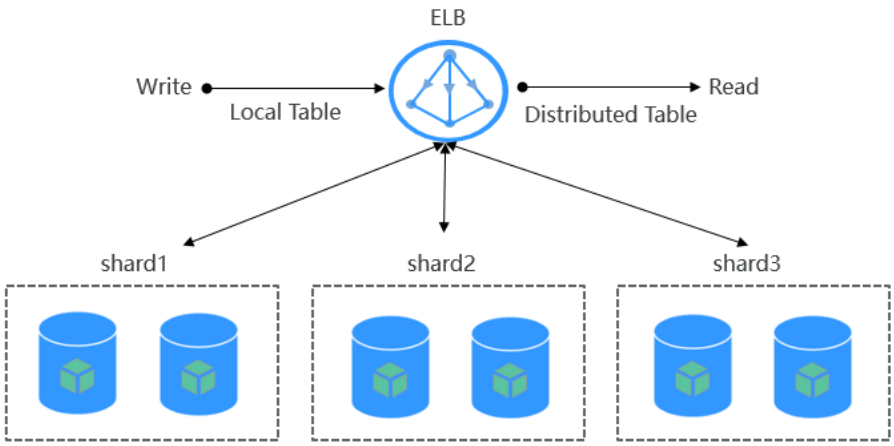


- Arquitectura de despliegue de HA

  MRS utiliza la arquitectura de implementación de alta disponibilidad (HA) basada en ELB para distribuir automáticamente el tráfico de acceso de usuario a múltiples nodos de back-end, ampliando las capacidades de servicio a sistemas externos y mejorando la tolerancia a fallas. Como se muestra en **Figura 6-5**, cuando una aplicación de cliente solicita un clúster, se utiliza Elastic Load Balance (ELB) para distribuir el tráfico. Con el mecanismo de sondeo de ELB, los datos se escriben en tablas locales y se leen de tablas distribuidas en diferentes nodos. De esta manera, se garantiza la carga de lectura/escritura de datos y la alta disponibilidad de acceso a la aplicación.

  Después de aprovisionar el clúster de ClickHouse, cada nodo de instancia de ClickHouse del clúster corresponde a una réplica y dos réplicas forman una partición lógica. Por ejemplo, al crear una tabla ReplicatedMergeTree, puede especificar particiones para que los datos se sincronicen automáticamente entre dos réplicas en la misma partición.

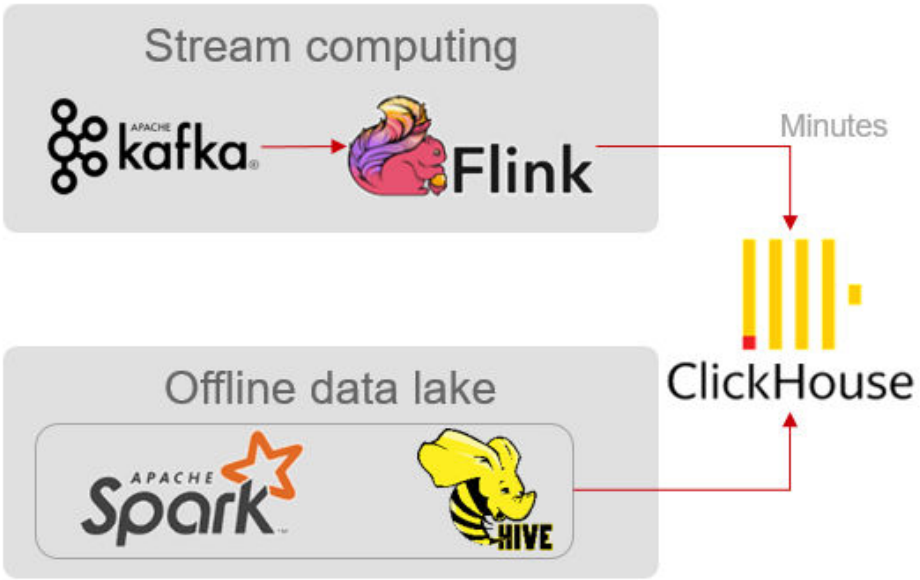
Figura 6-5 Arquitectura de despliegue de HA



Relación entre ClickHouse y otros componentes

ClickHouse depende de ZooKeeper para instalación y despliegue.

Las aplicaciones informáticas de flujo de Flink se utilizan para generar datos de informe comunes (tablas planas de detalle) y escribir los datos de informe en ClickHouse en tiempo casi real. Los trabajos de Hive/Spark se utilizan para generar datos de informes comunes (tablas planas de detalle) e importar los datos por lotes a ClickHouse.



NOTA

Actualmente ClickHouse no admite la interconexión con Kafka en modo normal ni HDFS en modo de seguridad.

6.5 DBService

6.5.1 Principios básicos de DBService

Descripción

DBService es un sistema de almacenamiento HA para bases de datos relacionales, que es aplicable al escenario donde se necesita almacenar una pequeña cantidad de datos (aproximadamente 10 GB), por ejemplo, metadatos de componentes. DBService solo puede ser utilizado por componentes internos de un clúster y proporciona funciones de almacenamiento, consulta y eliminación de datos.

DBService es un componente básico de un clúster. Componentes como Hive, Hue, Oozie, Loader y Redis, y Loader almacenan sus metadatos en DBService y proporcionan las funciones de copia de respaldo y restauración de metadatos mediante DBService.

Arquitectura de DBService

DBService en el clúster funciona en modo activo/en espera. Se implementan dos instancias de DBServer y cada instancia contiene tres módulos: HA, Database y FloatIP.

Figura 6-6 muestra la arquitectura lógica de DBService.

Figura 6-6 Arquitectura de DBService

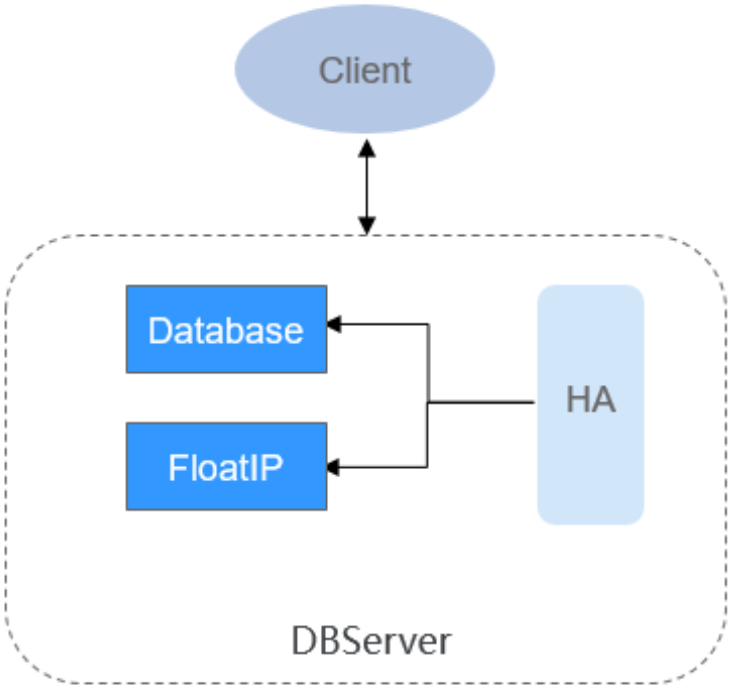


Tabla 6-4 describe los módulos mostrados en **Figura 6-6**

Tabla 6-4 Descripción del módulo

Nombre	Descripción
HA	Módulo de gestión de HA. El DBServer activo/en espera utiliza el módulo de HA para la gestión.
Database	Módulo de base de datos. Este módulo almacena los metadatos del módulo de Cliente.
FloatIP	Dirección IP flotante que proporciona la función de acceso externamente. Solo está habilitado en la instancia activa de DBServer y es utilizado por el módulo de Client para acceder a la base de datos.
Client	Cliente que utiliza el componente de DBService, que se despliega en el nodo de instancia del componente. El cliente se conecta a la base de datos mediante FloatIP y, a continuación, realiza operaciones de adición, eliminación y modificación de metadatos.

6.5.2 Relación entre DBService y otros componentes

DBService es un componente básico de un clúster. Los componentes como Hive, Hue, Oozie, Loader, Metadata, Redis y Loader almacenan sus metadatos en DBService y proporcionan las funciones de copia de respaldo y restauración de metadatos mediante DBService.

6.6 Flink

6.6.1 Principios básicos de Flink

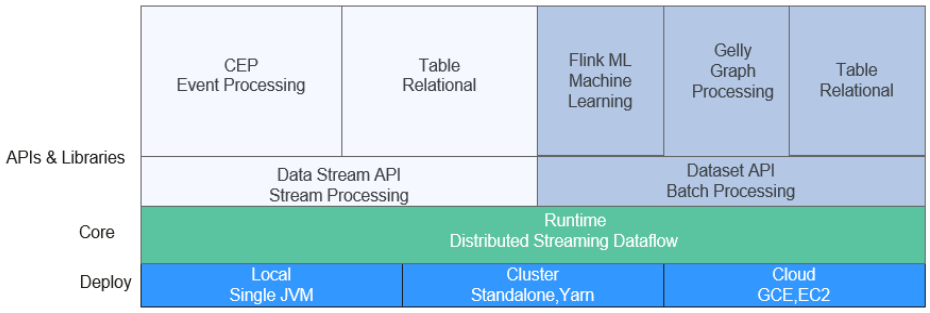
Descripción

Flink es un marco de computación unificado que soporta tanto el procesamiento por lotes como el procesamiento de flujo. Proporciona un motor de procesamiento de datos de flujo que admite la distribución de datos y la computación en paralelo. Flink cuenta con procesamiento de flujo y es un motor de procesamiento de flujo de código abierto superior en la industria.

Flink proporciona procesamiento de datos de canalización de alta concurrencia, latencia de nivel de milisegundos y alta confiabilidad, lo que lo hace extremadamente adecuado para el procesamiento de datos de baja latencia.

Figura 6-7 muestra la pila de tecnología de Flink.

Figura 6-7 Pila de tecnología de Flink



Flink proporciona las siguientes características en la versión actual:

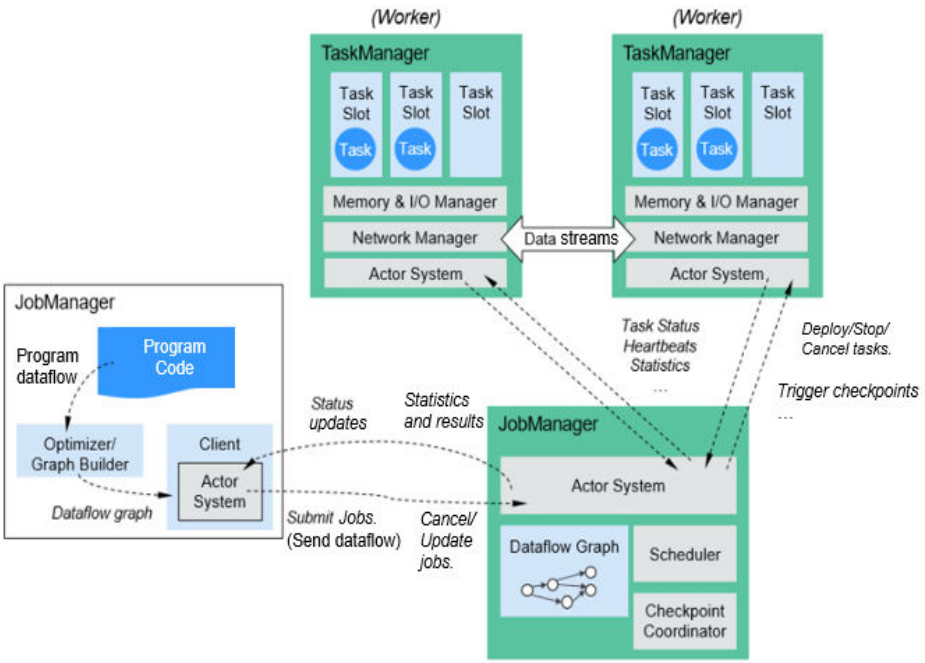
- DataStream
- Checkpoint
- Window
- Job Pipeline
- Configuration Table

Otras características son heredadas de la comunidad de código abierto y no son mejoradas. Para obtener más información, visite **https://ci.apache.org/projects/flink/flink-docs-release-1.12/**.

Arquitectura de Flink

Figura 6-8 muestra la arquitectura de Flink.

Figura 6-8 Arquitectura de Flink



Como se muestra en la figura anterior, todo el sistema de Flink consta de tres partes:

- Client

  Flink client se utiliza para enviar trabajos (trabajos de streaming) a Flink.

- TaskManager

  TaskManager es un nodo de ejecución de servicio de Flink. Ejecuta tareas específicas. Un sistema de Flink puede tener múltiples TaskManagers. Estos TaskManagers son equivalentes entre sí.

- JobManager

  JobManager es un nodo de gestión de Flink. Gestiona todas las TaskManagers y programa las tareas enviadas por los usuarios a TaskManagers específicos. En el modo

de alta disponibilidad (HA), se despliega múltiples JobManagers. Entre estos JobManagers se selecciona uno como el JobManager activo, y los otros están en espera.

Para obtener más información sobre la arquitectura de Flink, visite **https://ci.apache.org/projects/flink/flink-docs-master/docs/concepts/flink-architecture/**.

Principios de Flink

- **Stream & Transformation & Operator**

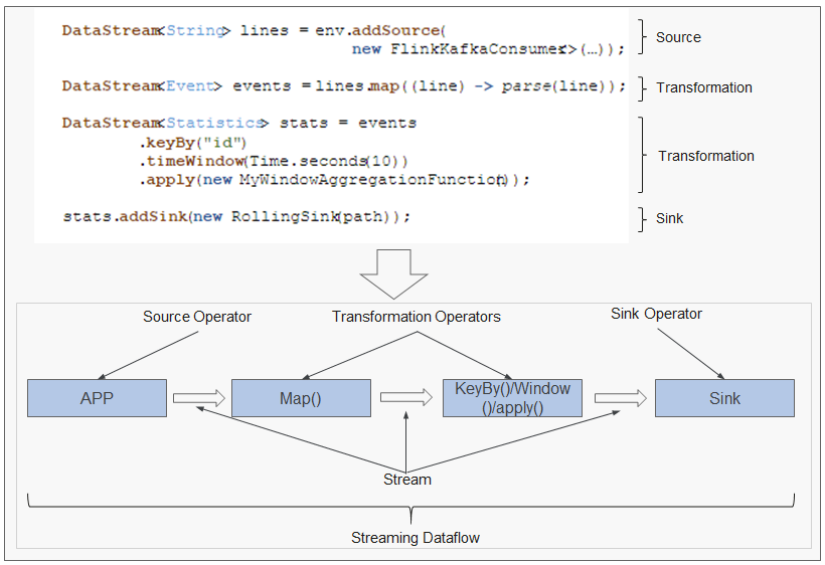
  Un programa Flink consta de dos bloques de construcción: stream and transformation.

  a. Conceptualmente, un stream es un flujo (potencialmente interminable) de registros de datos, y transformation es una operación que toma uno o más streams como entrada, y produce uno o más streams de salida como resultado.

  b. Cuando se ejecuta un programa Flink, se asigna a un streaming dataflow. Un streaming dataflow consiste en un grupo de streams y transformation operator. Cada dataflow comienza con uno o más source operators y termina en uno o más sink operators. Un dataflow se asemeja a un gráfico acíclico dirigido (DAG).

  Figura 6-9 muestra el streaming dataflow al que se mapea un programa Flink.

  Figura 6-9 Ejemplo de Flink DataStream

  Como se muestra en el **Figura 6-9**, **FlinkKafkaConsumer** es un operador de origen; Map, KeyBy, TimeWindow y Apply son transformation operators; RollingSink es un sink operator.

- **Pipeline Dataflow**

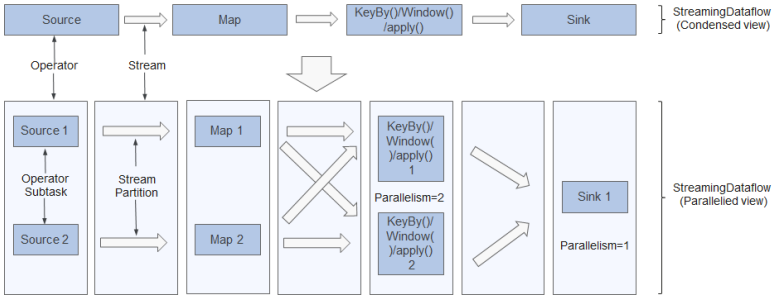
  Las aplicaciones en Flink pueden ejecutarse en modo paralelo o distribuido. Un stream puede dividirse en una o más particiones de stream, y un operator puede dividirse en múltiples operator subtasks.

  El ejecutor de streams y operators se optimiza automáticamente en función de la densidad de los operadores ascendentes y descendentes.

  - Los operadores con baja densidad no se pueden optimizar. Cada operator subtask se ejecuta por separado en diferentes subprocesos. El número de operator subtasks es el paralelismo de ese operador en particular. El paralelismo (el número total de

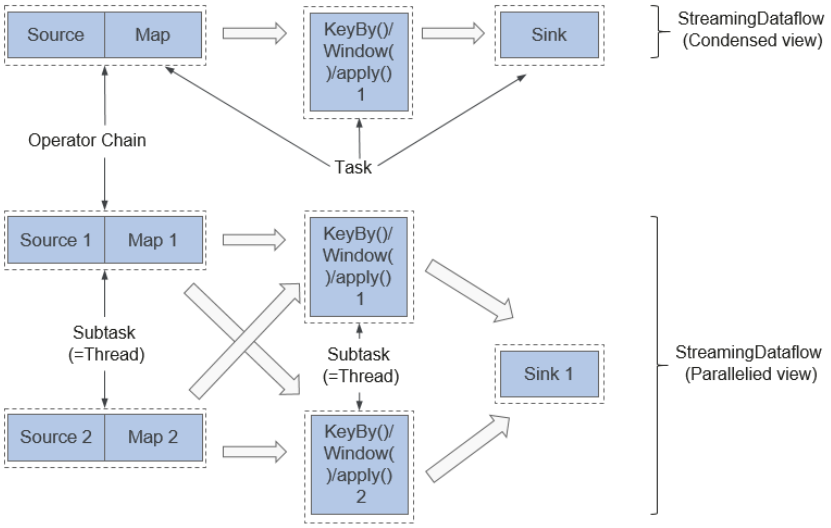
particiones) de un stream es el de su operator productor. Diferentes operators del mismo programa pueden tener diferentes niveles de paralelismo, como se muestra en **Figura 6-10**.

Figura 6-10 Operator



- Los operadores con alta densidad pueden ser optimizados. Flink encadena operator subtasks en una tarea, es decir, un operator chain. Cada operator chain es ejecutada por un subproceso en TaskManager como se muestra en **Figura 6-11**.

Figura 6-11 Operator chain



  - En la parte superior de **Figura 6-11**, los operadores de Source y Map condensados están encadenados en un Operator Chain, es decir, un operador más grande. El Operator Chain, el KeyBy y el Sink representan respectivamente a un operador y están conectados entre sí a través de streams. Cada operador corresponde a un task durante la ejecución. Es decir, hay tres tasks en la parte superior.
  - En la parte inferior de **Figura 6-11** cada task, excepto Sink, es paralela en dos subtasks. El paralelismo del operador de Sink es uno.

Características clave

- Procesamiento de flujos

  El motor de procesamiento de flujo en tiempo real cuenta con alto rendimiento, alto rendimiento y baja latencia, que puede proporcionar capacidad de procesamiento en milisegundos.

- Diferentes gestión de estado

  La aplicación de procesamiento de flujo necesita almacenar los eventos recibidos o el resultado intermedio en un cierto período de tiempo para el acceso y procesamiento posteriores en un cierto punto de tiempo. Flink ofrece diversas funciones para la gestión de estado, que incluye:

  - Múltiples tipos de estado básicos: Flink proporciona varios estados para estructuras de datos, como ValueState, ListState y MapState. Los usuarios pueden seleccionar el tipo de estado más eficiente y adecuado basado en el modelo de servicio.
  - State Backend rico: State Backend gestiona el estado de las aplicaciones y realiza operaciones de Checkpoint según sea necesario. Flink proporciona diferentes State Backends. State se puede almacenar en la memoria o RocksDB, y soporta el mecanismo de Checkpoint asincrónico e incremental.
  - Consistencia de estado de una sola vez: Las capacidades de Checkpoint y recuperación de fallas de Flink garantizan que el estado de la aplicación de las tareas sea consistente antes y después de que se produzca un falla. Flink admite salida transaccional para algunos dispositivos de almacenamiento específicos. De esta manera, se puede garantizar una salida exactamente una vez incluso cuando se produce una falla.

- Varias semánticas de tiempo

  El tiempo es una parte importante de las aplicaciones de procesamiento de flujo. Para aplicaciones de procesamiento de flujo en tiempo real, son muy comunes operaciones tales como agregación de ventanas, detección y emparejamiento basado en semántica de tiempo. Flink proporciona varias semánticas de tiempo.

  - Event-time: La marca de tiempo proporcionada por el evento se utiliza para el cálculo, facilitando el procesamiento de los eventos que llegan a una secuencia aleatoria o llegan tarde.
  - Watermark: Flink introduce el concepto de Watermark para medir el desarrollo del tiempo de eventos. Watermkark también proporciona una garantía flexible para balancear la latencia del procesamiento y la integridad de los datos. Cuando se procesan flujos de eventos con Watermark, Flink proporciona múltiples opciones de procesamiento si los datos llegan después del cálculo, por ejemplo, redirigir datos (salida lateral) o actualizar el resultado del cálculo.
  - Se admite Processing-time y Ingestion-time.
  - Ventana de streaming altamente flexible: Flink admite la ventana de tiempo, la ventana de conteo, la ventana de sesión y la ventana personalizada basada en datos. Puede personalizar las condiciones de activación para implementar el modo de cálculo de streaming complejo.

- Mecanismo de tolerancia a fallas

  En un sistema distribuido, si una única tarea o nodo se descompone o está defectuoso, toda la tarea puede fallar. Flink proporciona un mecanismo de tolerancia a fallas a nivel de tarea, que garantiza que los datos del usuario no se pierdan cuando se produzca una excepción en una tarea y se pueda restaurar automáticamente.

  - Checkpoint: Flink implementa la tolerancia a fallas basada en checkpoint. Los usuarios pueden personalizar la política de checkpoint para toda la tarea. Cuando se produce un error en una tarea, la tarea se puede restaurar al estado del último checkpoint y los datos más recientes después de que la instantánea se reenvíe desde el origen de datos.
  - Savepoint: Un savepoint es una instantánea consistente del estado de la aplicación. El mecanismo de savepoint es similar al de checkpoint. Sin embargo, el mecanismo

de savepoint necesita ser activado manualmente. El mecanismo de savepoint garantiza que la información de estado de la aplicación de flujo actual no se pierda durante la actualización o migración de tareas, lo que facilita la suspensión y recuperación de tareas en cualquier punto de tiempo.

- Flink SQL

  Table APIs y SQL usan Apache Calcite para analizar, verificar y optimizar consultas. Table APIs y SQL se pueden integrar sin problemas con DataStream y DataSet APIs, y admiten funciones escalares definidas por el usuario, funciones de agregación y funciones de valor de tabla. La definición de aplicaciones tales como análisis de datos y ETL se simplifica. En el ejemplo de código siguiente se muestra cómo utilizar sentencias de Flink SQL para definir una aplicación de recuento que registra los tiempos de sesión.

  ```
  SELECT userId, COUNT(*)
  FROM clicks
  GROUP BY SESSION(clicktime, INTERVAL '30' MINUTE), userId
  ```

  Para obtener más información acerca de Flink SQL, consulte **https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html**.

- CEP en SQL

  Flink permite a los usuarios representar resultados de consultas de procesamiento de eventos complejos (CEP) en SQL para la coincidencia de patrones y evaluar flujos de eventos en Flink.

  CEP SQL se implementa a través de la sintaxis de SQL **MATCH_RECOGNIZE**. La cláusula **MATCH_RECOGNIZE** es compatible con Oracle SQL desde Oracle Database 12c y se utiliza para indicar la coincidencia de patrones de eventos en SQL. El siguiente es un ejemplo de CEP SQL:

  ```
  SELECT T.aid, T.bid, T.cid
  FROM MyTable
      MATCH_RECOGNIZE (
        PARTITION BY userid
        ORDER BY proctime
        MEASURES
          A.id AS aid,
          B.id AS bid,
          C.id AS cid
        PATTERN (A B C)
        DEFINE
          A AS name = 'a',
          B AS name = 'b',
          C AS name = 'c'
      ) AS T
  ```

6.6.2 Solución de Flink HA

Solución de Flink HA

Un clúster de Flink solo tiene un JobManager. Esto tiene el riesgo de un punto único de fallas (SPOF). Hay tres modos de Flink: Flink On Yarn, Flink Standalone y Flink Local. Los modos de Flink On Yarn y Flink Standalone se basan en clústeres y el modo de Flink Local se basa en un solo nodo. Flink On Yarn y Flink Standalone proporcionan un mecanismo de HA. Con este mecanismo, puede recuperar el JobManager de fallos y, por lo tanto, eliminar los riesgos de SPOF. Esta sección describe el mecanismo de HA del Flink On Yarn.

Flink admite el modo de HA y la recuperación de excepciones de trabajo que dependen en gran medida de ZooKeeper. Si desea habilitar las dos funciones, configure ZooKeeper en el archivo **flink-conf.yaml** de antemano de la siguiente manera:


```
high-availability: zookeeper
high-availability.zookeeper.quorum:  ZooKeeper IP address:2181
high-availability.storageDir: hdfs:///flink/recovery
```

Flink On Yarn

JobManager de Flink y ApplicationMaster de Yarn están en el mismo proceso. ResourceManager de Yarn monitorea ApplicationMaster. Si ApplicationMaster es anormal, Yarn lo reinicia y restaura todos los metadatos de JobManager desde HDFS. Durante la recuperación, las tareas existentes no se pueden ejecutar y no se pueden enviar nuevas tareas. ZooKeeper almacena metadatos de JobManager, como información sobre jobs, para que los utilice el nuevo JobManager. Un error de TaskManager es escuchado y procesado por el mecanismo de DeathWatch de Akka en JobManager. Cuando un TaskManager falla, se solicita de nuevo un contenedor de Yarn y se crea un TaskManager.

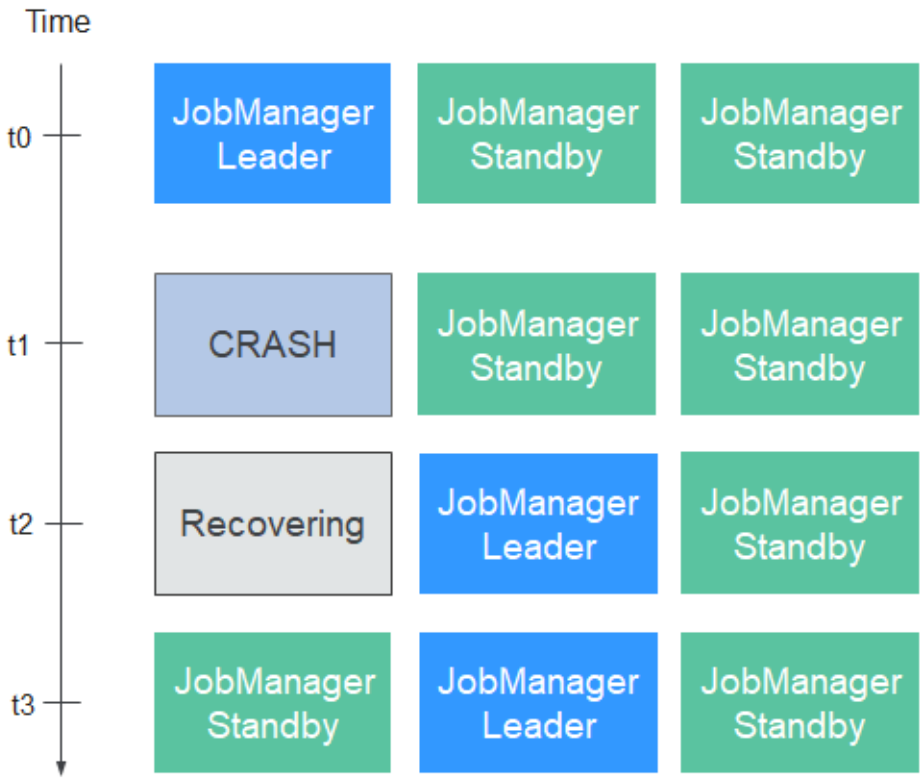
Para obtener más información sobre la solución de HA de Flink en YARN, visite:

http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

Standalone

En el modo de standalone, se pueden iniciar varios JobManagers y ZooKeeper elige uno como leader de JobManager. En este modo, hay un JobManager líder y varios JobManagers en espera. Si el líder JobManager falla, un JobManager de reserva se hace cargo de la dirección. **Figura 6-12** muestra el proceso de una conmutación de JobManager líder/en espera.

Figura 6-12 Proceso de conmutación



Restauración de TaskManager

Un error de TaskManager es escuchado y procesado por el mecanismo de DeathWatch de Akka en JobManager. Si el TaskManager falla, el JobManager crea un TaskManager y migra servicios al TaskManager creado.

Restauración de JobManager

JobManager de Flink y ApplicationMaster de Yarn están en el mismo proceso. ResourceManager de Yarn monitorea ApplicationMaster. Si ApplicationMaster es anormal, Yarn lo reinicia y restaura todos los metadatos de JobManager desde HDFS. Durante la recuperación, las tareas existentes no se pueden ejecutar y no se pueden enviar nuevas tareas.

Restauración de Jobs

Si desea restaurar trabajos, asegúrese de que la política de inicio está configurada en los archivos de configuración de Flink. Las políticas de reinicio admitidas son **fixed-delay**, **failure-rate** y **none**. Jobs sólo se pueden restaurar cuando la política está configurada en **fixed-delay** o **failure-rate**. Si la política de reinicio está configurada en **none** y el punto de control está configurado para trabajos, la política de reinicio se configura automáticamente en **fixed-delay** y el valor de **restart-strategy.fixed-delay.attempts** (que especifica el número de veces de reintentos) se configura en **Integer.MAX_VALUE**.

Para más detalles sobre las tres estrategias, visite el sitio web oficial de Flink en **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/task_failure_recovery.html**. Las estrategias de configuración son las siguientes:

```
restart-strategy: fixed-delay
restart-strategy.fixed-delay.attempts: 3
restart-strategy.fixed-delay.delay: 10 s
```

Jobs se restaurarán en los siguientes escenarios:

- Si un JobManager falla, todos sus jobs se detienen y se recuperarán después de crear y ejecutar otro JobManager.
- Si un TaskManager falla, todas las tareas en el TaskManager se detienen y se iniciarán hasta que haya recursos disponibles.
- Cuando una tarea de un job falla, el job se reinicia.

  NOTA

  Para obtener más información acerca de cómo configurar las estrategias de reinicio de job, consulte **https://ci.apache.org/projects/flink/flink-docs-release-1.12/ops/jobmanager_high_availability.html**.

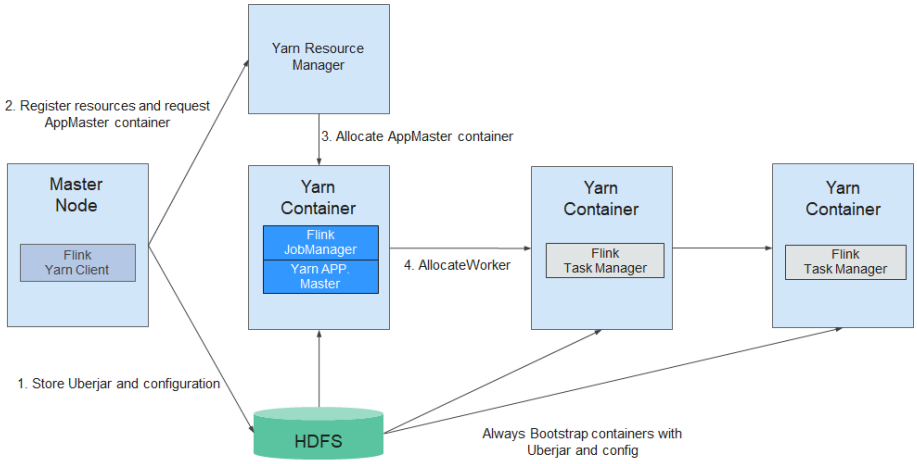
6.6.3 Relación entre Flink y otros componentes

Relación entre Flink y YARN

Flink admite el modo de gestión de clústeres basado en YARN. En este modo, Flink sirve como una aplicación de YARN y se ejecuta en YARN.

Figura 6-13 muestra el despliegue del clúster de Flink basado en YARN.

Figura 6-13 Despliegue de clúster de Flink basada en YARN



1. El Flink YARN Client primero comprueba si hay suficientes recursos para iniciar el clúster de YARN. En caso afirmativo, el Flink YARN client carga archivos de JAR y archivos de configuración a HDFS.
2. Flink YARN client se comunica con YARN ResourceManager para solicitar un container para iniciar el ApplicationMaster. Después de que todos los NodeManagers de YARN terminen de descargar el archivo de JAR y los archivos de configuración, el ApplicationMaster se inicia.
3. Durante el inicio, ApplicationMaster interactúa con YARN ResourceManager para solicitar el container para iniciar un TaskManager. Una vez que el container está listo, se inicia el proceso de TaskManager.
4. En el clúster de Flink YARN, el ApplicationMaster y Flink JobManager se ejecutan en el mismo container. El ApplicationMaster informa a cada TaskManager de la dirección de RPC del JobManager. Después de iniciar TaskManagers, se registran en el JobManager.
5. Después de que todos los TaskManagers se hayan registrado con el JobManager Flink se inicia en el clúster de YARN. A continuación, el Flink YARN client puede enviar Flink jobs al JobManager y Flink puede realizar la asignación, programación y cálculo para los jobs.

6.6.4 Funciones de código abierto mejoradas de Flink

6.6.4.1 Ventana

Función de código abierto mejorada: Ventana

Esta sección describe la ventana deslizante de Flink y proporciona el método de optimización de ventana deslizante. Para obtener más información sobre ventanas, visite el sitio web oficial en **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/stream/operators/windows.html**.

Introducción a ventana

Los datos de una ventana se guardan como resultados intermedios o como datos originales. Si realiza una operación de suma (**window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).sum**) en los datos de la ventana, solo se conservará el resultado intermedio. Si se utiliza una ventana personalizada

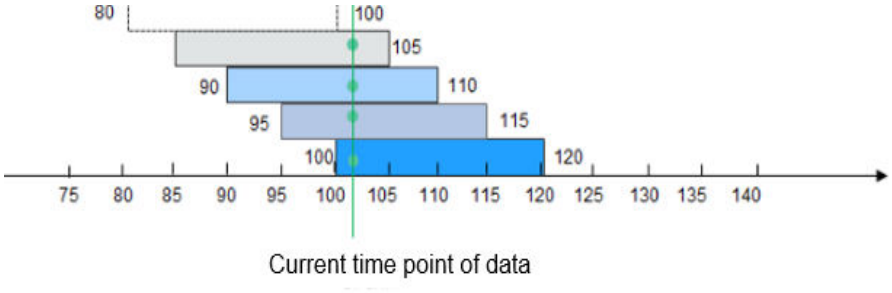
(**window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).apply(new UDF)**), se guardarán todos los datos originales de la ventana.

Si se utilizan **SlidingEventTimeWindow** y **SlidingProcessingTimeWindow** de ventanas personalizadas, los datos se guardan como múltiples copias de respaldo. Supongamos que la ventana se define de la siguiente manera:

```
window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).apply(new UDFWindowFunction)
```

Si llega un bloque de datos, se asigna a cuatro ventanas diferentes (20/5 = 4). Es decir, los datos se guardan como cuatro copias en la memoria. Cuando el tamaño de la ventana o el período de deslizamiento se establece en un valor grande, los datos se guardarán como copias excesivas, lo que causa redundancia.

Figura 6-14 Estructura original de una ventana



Si un bloque de datos llega al segundo 102, se asigna a las ventanas [85, 105), [90, 110), [95, 115) y [100, 120).

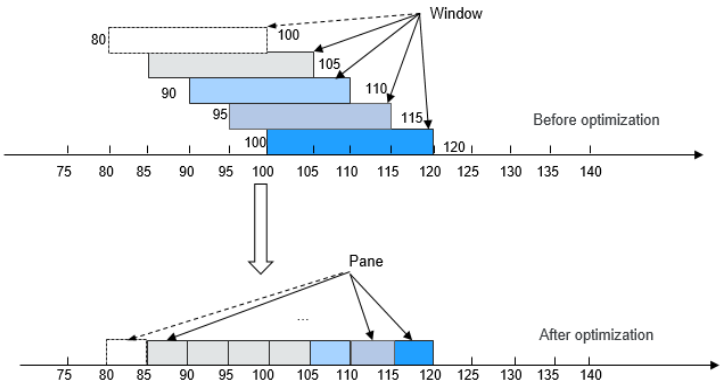
Optimización de Window

Como se mencionó en lo anterior, hay copias de datos excesivas cuando los datos originales se guardan en el SlidingEventTimeWindow y el SlidingProcessingTimeWindow. Para resolver este problema, se reestructura la ventana que almacena los datos originales, lo que optimiza el almacenamiento y reduce considerablemente el espacio de almacenamiento. El esquema de optimización de ventanas es el siguiente:

1. Utilice el período deslizante como una unidad para dividir una ventana en diferentes paneles.

   Una ventana consta de uno o varios paneles. Un panel es esencialmente un período deslizante. Por ejemplo, el período deslizante (es decir, el panel) de **window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds.of(5)))** dura 5 segundos. Si esta ventana oscila entre [100, 120), esta ventana se puede dividir en paneles [100, 105), [105, 110), [110, 115) y [115, 120).

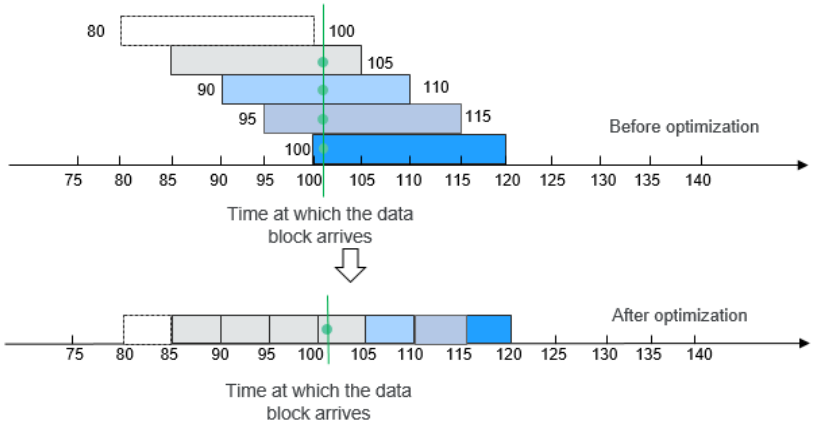
Figura 6-15 Optimización de ventanas



2. Cuando llega un bloque de datos, no se asigna a una ventana específica. En su lugar, Flink determina el panel al que pertenece el bloque de datos basándose en la marca de tiempo del bloque de datos y guarda el bloque de datos en el panel.

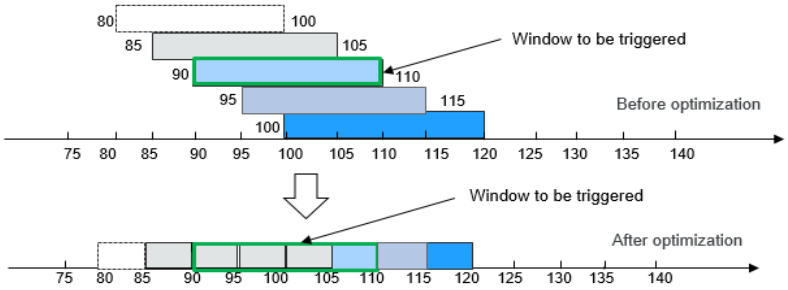
   Un bloque de datos sólo se guarda en un panel. En este caso, solo existe una copia de datos en la memoria.

Figura 6-16 Guardar datos en una ventana



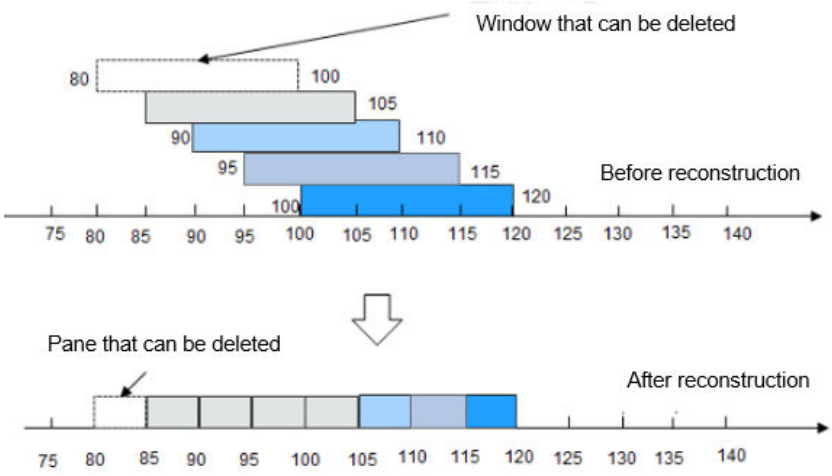
3. Para activar una ventana, compute todos los paneles contenidos en la ventana y combine todos estos paneles en una ventana completa.

Figura 6-17 Activar una ventana



4. Si no se requiere un panel, puede eliminarlo de la memoria.

Figura 6-18 Eliminar una ventana



Después de la optimización, la cantidad de copias de datos en la memoria y la instantánea se reduce considerablemente.

6.6.4.2 Job Pipeline

Función de código abierto mejorada: Job Pipeline

Generalmente, el código lógico relacionado con un servicio se almacena en un paquete de JAR grande, que se llama Fat JAR. Las desventajas de Fat JAR son las siguientes:

- Cuando la lógica de servicio se vuelve cada vez más compleja, el tamaño de Fat JAR aumenta.
- Fat Jar hace que la coordinación sea compleja. Los desarrolladores de todos los servicios están trabajando con la misma lógica de servicio. Aunque la lógica de servicio puede dividirse en varios módulos, todos los módulos están estrechamente acoplados entre sí. Si es necesario cambiar el requisito, es necesario volver a planificar todo el diagrama de flujo.

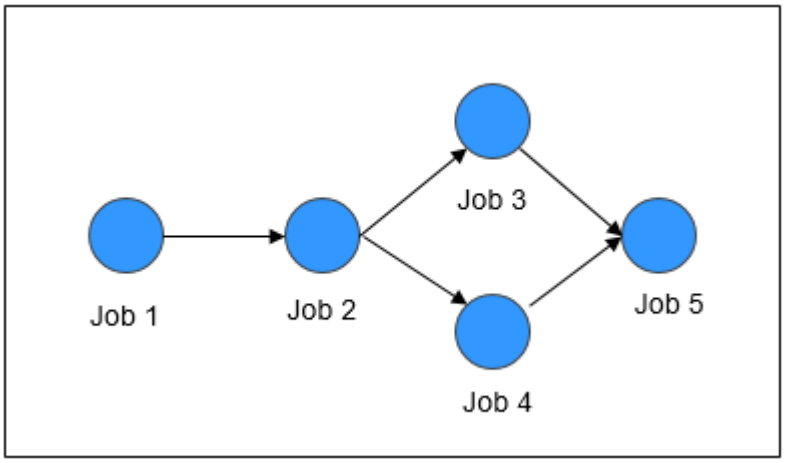
La división de puestos de trabajo se enfrenta a los siguientes problemas:

- La transmisión de datos entre trabajos se puede lograr utilizando Kafka. Por ejemplo, el trabajo A transmite datos al topic A en Kafka y, a continuación, el trabajo B y el trabajo C leen datos del topic A en Kafka. Esta solución es simple y fácil de implementar, pero la latencia es siempre superior a 100 ms.
- Los operadores se conectan mediante el protocolo TCP. En un entorno distribuido, los operadores pueden programarse en cualquier nodo y los servicios ascendentes y descendentes no pueden detectar la programación.

Job Pipeline

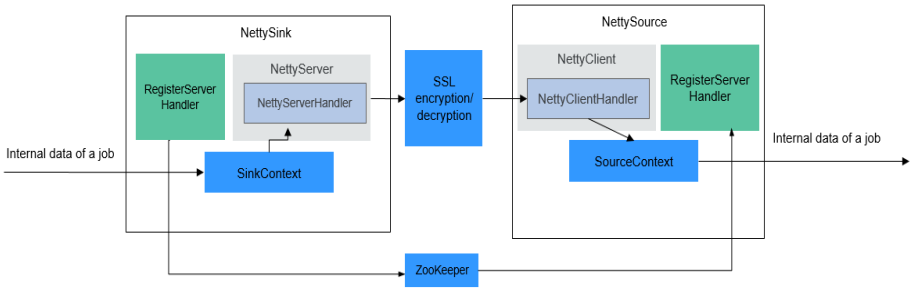
Un Pipeline consiste en múltiples jobs de Flink conectados a través de TCP. Los jobs ascendentes pueden enviar datos a los jobs descendentes. El diagrama de flujo sobre la transmisión de datos se denomina canalización de trabajos, como se muestra en **Figura 6-19**.

Figura 6-19 Job pipeline



Principios de Job Pipeline

Figura 6-20 Principios de Job pipeline



- NettySink y NettySource

  En un pipeline, los jobs ascendentes y descendentes se comunican entre sí a través de Netty. El operador Sink del job ascendente funciona como un server y el operador Source del job descendente funciona como un cliente. El operador Sink del job ascendente se denomina NettySink y el operador Source del job descendente se denomina NettySource.

- NettyServer y NettyClient

  NettySink funciona como el servidor de Netty. En NettySink, NettyServer logra la función de un servidor. NettySource funciona como el cliente de Netty. En el NettySource, NettyClient cumple la función de un cliente.

- Publicador

  El job que envía datos a jobs descendentes a través de NettySink se denomina publicador.

- Suscriptor

  El job que recibe datos de jobs ascendentes a través de NettySource se denomina suscriptor.

- RegisterServer

  RegisterServer es la memoria de terceros que almacena la dirección IP, el número de puerto y la información de simultaneidad de NettyServer.

- La arquitectura general exterior-interior es la siguiente:
  - NettySink->NettyServer->NettyServerHandler
  - NettySource->NettyClient->NettyClientHandler

Funciones de Job Pipeline

- **NettySink**

  NettySink consta de los siguientes módulos principales:

  - RichParallelSinkFunction

    NettySink hereda el RichParallelSinkFunction y los atributos de los operadores de Sink. La API de RichParallelSinkFunction implementa las siguientes funciones:

    - Inicia el operador de NettySink.
    - Ejecuta el operador de NettySink y recibe datos del operador ascendente.
    - Cancela la ejecución de los operadores de NettySink.

    La siguiente información se puede obtener utilizando el atributo de RichParallelSinkFunction:

    - subtaskIndex acerca de la concurrencia de cada operador de NettySink.
    - Concurrencia del operador de NettySink.
  - RegisterServerHandler

    RegisterServerHandler interactúa con el componente de RegisterServer y define las siguientes API:

    - **start();**: Inicia el RegisterServerHandler y establece un contacto con el RegisterServer de terceros.
    - **createTopicNode();**: Crea un nodo de topic.
    - **register();**: Registra información como la dirección IP, el número de puerto y la simultaneidad en el nodo de topic.
    - **deleteTopicNode();**: Elimina un nodo de topic.
    - **unregister();**: Borra la información de registro.
    - **query();**: Consulta información de registro.
    - **isExist();**: Verifica que existe una información específica.
    - **shutdown();**: Desactiva el RegisterServerHandler y se desconecta del RegisterServer de terceros.

NOTA

- API de RegisterServerHandler permite a ZooKeeper trabajar como el handler de RegisterServer. Puede personalizar su handler según sea necesario. La información se almacena en el ZooKeeper de la siguiente forma:

```
Namespace
|---Topic-1
  |---parallel-1
  |---parallel-2
  |....
  |---parallel-n
|---Topic-2
  |---parallel-1
  |---parallel-2
  |....
  |---parallel-m
|...
```

- La información sobre NameSpace se puede obtener de los siguientes parámetros del archivo **flink-conf.yaml**:

```
nettyconnector.registerserver.topic.storage: /flink/nettyconnector
```

- La autenticación simple de autenticación y capa de seguridad (SASL) entre ZookeeperRegisterServerHandler y ZooKeeper se implementa a través del marco de Flink.
- Asegúrese de que cada job tiene un topic único. De lo contrario, la relación de suscripción puede ser poco clara.
- Al invocar a **shutdown()**, ZookeeperRegisterServerHandler elimina la información de registro sobre la concurrencia actual y, a continuación, intenta eliminar el nodo de topic. Si el nodo de topic no está vacío, se cancelará la eliminación, ya que no se ha terminado toda la concurrencia.

- NettyServer

  NettyServer es el núcleo del operador de NettySink, cuya función principal es crear un NettyServer y recibir solicitudes de conexión de NettyClient. Utilice NettyServerHandler para enviar datos recibidos de operadores ascendentes de un mismo job. El número de puerto y la subred de NettyServer deben configurarse en el archivo **flink-conf.yaml**.

  - Rango de puertos

    ```
    nettyconnector.sinkserver.port.range: 28444-28943
    ```

  - Subred

    ```
    nettyconnector.sinkserver.subnet: 10.162.222.123/24
    ```

    NOTA

    El parámetro **nettyconnector.sinkserver.subnet** se establece en la subred (dirección IP del servicio) del cliente Flink de forma predeterminada. Si el cliente y TaskManager no están en la misma subred, puede producirse un error. Por lo tanto, debe establecer manualmente este parámetro en la subred (dirección IP del servicio) de TaskManager.

- NettyServerHandler

  El handler permite la interacción entre NettySink y suscriptores. Después de que NettySink recibe mensajes, el handler envía estos mensajes. Para garantizar la seguridad de la transmisión de datos, este canal se encripta mediante SSL. El **nettyconnector.ssl.enabled** configura si se habilita la encriptación sw SSL. La encriptación de SSL solo se habilita cuando **nettyconnector.ssl.enabled** se establece en **true**.

- **NettySource**

  NettySource consta de los siguientes módulos principales:

- RichParallelSourceFunction

  NettySource hereda RichParallelSinkFunction y atributos de los operadores de Source. La API de RichParallelSourceFunction implementa las siguientes funciones:

  - Inicia el operador de NettySink.
  - Ejecuta el operador de NettySink, recibe datos de los suscriptores e inyecta los datos a los jobs.
  - Cancela la ejecución de operadores de Source.

  La siguiente información se puede obtener utilizando el atributo RichParallelSourceFunction:

  - subtaskIndex acerca de la concurrencia de cada operador de NettySource.
  - Concurrencia del operador de NettySource.

  Cuando el operador del NettySource entra en la etapa de ejecución, se monitoriza el estado del NettyClient. Una vez que ocurre la anomalía, NettyClient se reinicia y se vuelve a conectar a NettyServer evitando la confusión de datos.

- RegisterServerHandler

  RegisterServerHandler de NettySource tiene una función similar a la del RegisterServerHandler de NettySink. Obtiene la dirección IP, número de puerto e información de operadores simultáneos de cada trabajo suscrito obtenido en el operador de NettySource.

- NettyClient

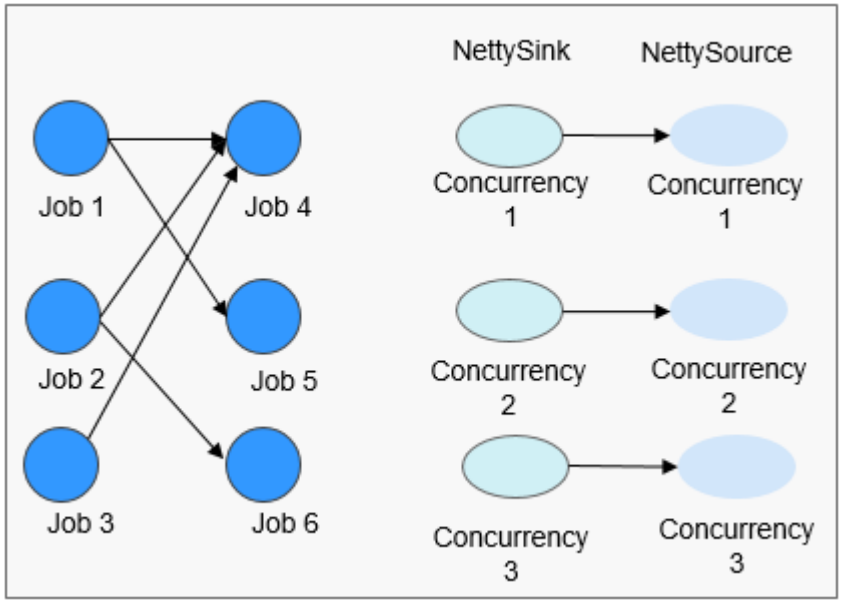
  NettyClient establece una conexión con NettyServer y utiliza NettyClientHandler para recibir datos. Cada operador de NettySource debe tener un nombre único (especificado por el usuario). NettyServer determina si cada cliente proviene de NettySources diferentes basados en nombres únicos. Cuando se establece una conexión entre NettyClient y NettyServer, NettyClient se registra con NettyServer y el nombre de NettySource de NettyClient se transfiere a NettyServer.

- NettyClientHandler

  El NettyClientHandler permite la interacción con los publicadores y otros operadores del job. Cuando se reciben mensajes, NettyClientHandler transfiere estos mensajes al job. Para garantizar la transmisión segura de los datos, NettySink está habilitado la encriptación de SSL para la comunicación con ellos. El encriptación SSL solo se habilita cuando SSL está habilitado y **nettyconnector.ssl.enabled** está establecido en **true**.

La relación entre los trabajos puede ser de muchos a muchos. La concurrencia entre cada operador NettySink y NettySource es de uno a varios, como se muestra en **Figura 6-21**.

Figura 6-21 Diagrama de relaciones



6.6.4.3 Stream SQL Join

Función de código abierto mejorada: Stream SQL Join

Flink's Table API&SQL es una API de consulta integrada para Scala y Java que permite la composición de consultas de operadores relacionales como selección, filtrado y unión de una manera intuitiva. Para obtener más información sobre Table API & SQL, visite el sitio web oficial en **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/table/index.html**.

Introducción a Stream SQL Join

SQL Join se utiliza para consultar datos basados en la relación entre columnas en dos o más tablas. Flink Stream SQL Join le permite unir dos tablas de streaming y consultar resultados de ellas. Se admiten consultas similares a las siguientes:

```
SELECT o.proctime, o.productId, o.orderId, s.proctime AS shipTime
FROM Orders AS o
JOIN Shipments AS s
ON o.orderId = s.orderId
AND o.proctime BETWEEN s.proctime AND s.proctime + INTERVAL '1' HOUR;
```

Actualmente, Stream SQL Join debe realizarse dentro de una ventana especificada. La operación de unión para datos dentro de la ventana requiere al menos un predicado de unión equivalente y una condición de unión que limite el tiempo en ambos lados. Tal condición puede definirse mediante dos predicados de rango (<, <=, >=, >) apropiados, un predicado de **BETWEEN** o un único predicado de igualdad que compara el mismo tipo de atributos de tiempo (tales como tiempo de procesamiento o tiempo de evento) de ambas tablas de entrada.

El siguiente ejemplo hará el join de todos los pedidos con sus envíos correspondientes si el pedido fue enviado cuatro horas después de que se recibió el pedido.

```
SELECT *
FROM Orders o, Shipments s
WHERE o.id = s.orderId AND
o.ordertime BETWEEN s.shiptime - INTERVAL '4' HOUR AND s.shiptime
```


NOTA

1. La unión de Stream SQL sólo admite inner join.
2. La cláusula **ON** debe incluir una condición de unión igual.
3. Los atributos de tiempo solo admiten el tiempo de procesamiento y el tiempo de evento.
4. La condición de ventana solo admite el intervalo de tiempo delimitado, por ejemplo **o.proctime BETWEEN s.proctime - INTERVAL '1' HOUR AND s.proctime + INTERVAL '1' HOUR**. El rango ilimitado, tal como **o.** proctime > s.proctime no es compatible. Se debe incluir el atributo **proctime** de dos flujos. **o.proctime BETWEEN proctime () AND proctime () + 1** no es compatible.

6.6.4.4 Flink CEP en SQL

Flink CEP en SQL

Flink permite a los usuarios representar resultados de consultas de procesamiento de eventos complejos (CEP) en SQL para la coincidencia de patrones y evaluar secuencias de eventos en motores de Flink.

Sintaxis de consulta SQL

CEP SQL se implementa a través de la sintaxis de SQL **MATCH_RECOGNIZE**. La cláusula **MATCH_RECOGNIZE** es compatible con Oracle SQL desde Oracle Database 12c y se utiliza para indicar la coincidencia de patrones de eventos en SQL. Apache Calcite también soporta la cláusula **MATCH_RECOGNIZE**.

Flink utiliza Calcite para analizar los resultados de las consultas SQL. Por lo tanto, esta operación cumple con la sintaxis de Apache Calcite.

```
MATCH_RECOGNIZE (
      [ PARTITION BY expression [, expression ]* ]
      [ ORDER BY orderItem [, orderItem ]* ]
      [ MEASURES measureColumn [, measureColumn ]* ]
      [ ONE ROW PER MATCH | ALL ROWS PER MATCH ]
      [ AFTER MATCH
            ( SKIP TO NEXT ROW
            | SKIP PAST LAST ROW
            | SKIP TO FIRST variable
            | SKIP TO LAST variable
            | SKIP TO variable )
      ]
      PATTERN ( pattern )
      [ WITHIN intervalLiteral ]
      [ SUBSET subsetItem [, subsetItem ]* ]
      DEFINE variable AS condition [, variable AS condition ]*
      )
```

Los elementos sintácticos de la cláusula **MATCH_RECOGNIZE** se definen de la siguiente manera:

-PARTITION BY (Opcional): define columnas de partición. Esta cláusula es opcional. Si este parámetro no está definido, se usa el paralelismo 1.

-ORDER BY (Opcional): define la secuencia de eventos en un flujo de datos. La cláusula **ORDER BY** es opcional. Si se omite, se utiliza la ordenación no determinista. Dado que el orden de los eventos es importante en la coincidencia de patrones, esta cláusula debe especificarse en la mayoría de los casos.

-MEASURES (Opcional): especifica el valor de atributo del evento correctamente coincidente.

-ONE ROW PER MATCH | ALL ROWS PER MATCH (Opcional): define cómo generar el resultado. **ONE ROW PER MATCH** indica que solo se genera una fila para cada coincidencia. **ALL ROWS PER MATCH** indica que se genera una fila para cada evento coincidente.

-AFTER MATCH (Opcional): especifica la posición de inicio para el procesamiento después de que el siguiente patrón se coincida correctamente.

-PATTERN: define el patrón coincidente como una expresión regular. Los siguientes operadores se pueden utilizar en la cláusula **PATTERN**: operadores de unión, operadores cuantificadores (*, +, ?, {n}, {n,}, {n,m} y {,m}), operadores de rama (barra vertical |) y operadores diferenciales ('{- -}').

(Opcional): **-WITHIN**: genera una coincidencia de cláusula de patrón solo cuando la coincidencia se produce dentro del tiempo especificado.

-SUBSET (Opcional): combina una o más variables asociadas definidas en la cláusula **DEFINE**.

-DEFINE: especifica la condición de Boolean, que define las variables utilizadas en la cláusula **PATTERN**.

Además, la cláusula **MATCH_RECOGNIZE** admite las siguientes funciones:

-MATCH_NUMBER(): se utiliza en la cláusula **MEASURES** para asignar el mismo número a cada fila que coincida correctamente.

-CLASSIFIER(): se utiliza en la cláusula **MEASURES** para indicar la asignación entre filas y variables coincidentes.

-FIRST() y **LAST()**: se utiliza en la cláusula **MEASURES** para devolver el valor de la expresión evaluada en la primera o la última fila del conjunto de filas asignado a la variable de esquema.

-NEXT() y **PREV()**: se utilizan en la cláusula **DEFINE** para evaluar una expresión utilizando la fila anterior o siguiente de una partición.

Palabras clave **-RUNNING** y **FINAL**: se utilizan para determinar la semántica necesaria para la agregación. **RUNNING** se puede usar en las cláusulas **MEASURES** y **DEFINE** mientras que **FINAL** solo se puede usar en la cláusula **MEASURES**.

- Funciones agregadas (**COUNT**, **SUM**, **AVG**, **MAX**, **MIN**): se utiliza en las cláusulas **MEASURES** y **DEFINE**.

Ejemplo de consulta

La siguiente consulta encuentra el patrón en forma de V en el flujo de datos de precios de acciones.

```
SELECT *
    FROM MyTable
    MATCH_RECOGNIZE (
      ORDER BY rowtime
      MEASURES
        STRT.name as s_name,
        LAST(DOWN.name) as down_name,
        LAST(UP.name) as up_name
      ONE ROW PER MATCH
      PATTERN (STRT DOWN+ UP+)
      DEFINE
        DOWN AS DOWN.v < PREV(DOWN.v),
```



```
        UP AS UP.v > PREV(UP.v)
    )
```

En la siguiente consulta, la función agregada **AVG** se utiliza en la cláusula **MEASURES** de **SUBSET E** que consiste en variables relacionadas con A y C.

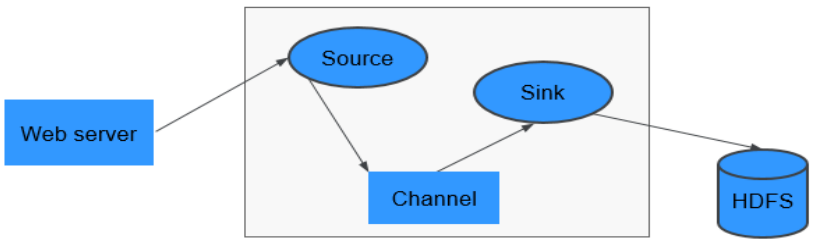
```
SELECT *
    FROM Ticker
    MATCH_RECOGNIZE (
      MEASURES
        AVG(E.price) AS avgPrice
      ONE ROW PER MATCH
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (A B+ C)
      SUBSET E = (A,C)
      DEFINE
        A AS A.price < 30,
        B AS B.price < 20,
        C AS C.price < 30
    )
```

6.7 Flume

6.7.1 Principios básicos de Flume

Flume es un sistema distribuido, confiable y de alta disponibilidad que admite la recopilación, agregación y transmisión masivas de log. Flume admite la personalización de varios emisores de datos en el sistema de log para la recopilación de datos. Además, Flume puede procesar datos y escribir datos en varios receptores de datos (personalizables). Un Flume-NG es una rama de Flume. Es simple, pequeño y fácil de implementar. La siguiente figura muestra la arquitectura básica del Flume-NG.

Figura 6-22 Arquitectura de Flume-NG



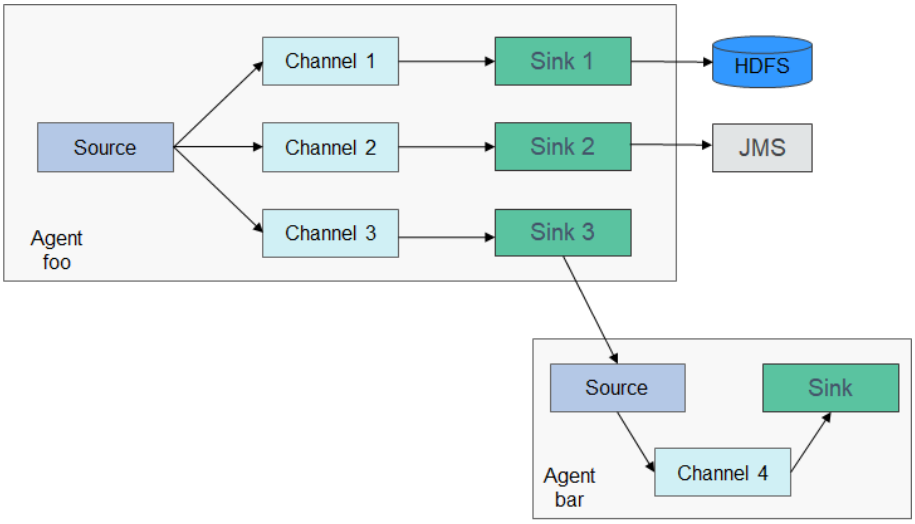
Un Flume-NG consiste en agents. Cada agent consta de tres componentes (source, channel, y sink). Se utiliza source para recibir datos. Se utiliza channel para transmitir datos. Se utiliza sink para enviar datos al siguiente extremo.

Tabla 6-5 Descripción del módulo

Módulo	Descripción
Source	Source recibe datos o genera datos usando un mecanismo especial, y coloca los datos en lotes en uno o más channels. Source puede funcionar en modo controlado por datos o sondeo. Los tipos de source típicos son los siguientes: ● Sources integradas con el sistema, como Syslog y Netcat ● Sources que generan eventos automáticamente, como Exec y SEQ ● Sources de IPC que se utilizan para la comunicación entre agents, como Avro Source debe estar asociada con al menos un channel.
Channel	Channel se utiliza para almacenar datos entre source y sink. Channel almacena en caché los datos de source y borra esos datos después de que sink envía los datos al channel del salto siguiente o al destino final. Diferentes channels proporcionan diferentes niveles de persistencia. ● Memory channel: no persistencia ● File channel: persistencia basada en registro de escritura previa (WAL) ● JDBC channel: persistencia implementada en base a la base de datos embebida Channel admite la función de transacción para garantizar operaciones secuenciales simples. Channel puede trabajar con sources y sinks de cualquier cantidad.
Sink	Sink envía datos al channel de salto siguiente o al destino final. Una vez completado, los datos transmitidos se eliminan del channel. Los tipos tipicos de sink son los siguientes: ● Sinks que envían datos de almacenamiento al destino final, como HDFS y HBase ● Sinks que se consumen automáticamente, como Null Sink ● IPC sinks utilizados para la comunicación entre Agents, como Avro Un sink debe estar asociado con un canal específico.

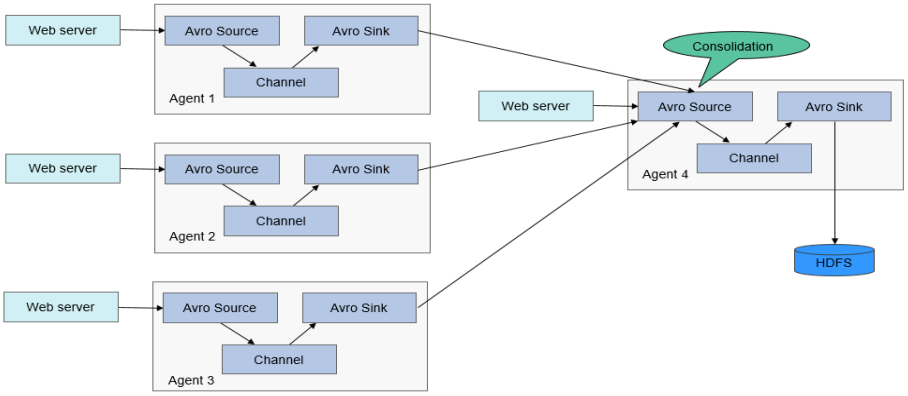
Como se muestra en **Figura 6-23**, un cliente de Flume puede tener múltiples sources, channels, y sinks.

Figura 6-23 Estructura de Flume



La confiabilidad de Flume depende de los cambios de transacción entre agents. Si el siguiente agent se descompone, el canal almacena datos de forma persistente y transmite datos hasta que el agente se recupera. La disponibilidad de Flume depende de los mecanismos integrados de balanceo de carga y migración por falla. Tanto el channel como el agent pueden configurarse con varias entidades entre las que pueden usar políticas de balanceo de carga. Cada agent es un proceso de máquina virtual de Java (JVM). Un servidor puede tener varios agents. Los nodos de recopilación (por ejemplo, Agents 1, 2, 3) procesan logs. Los nodos de agregación (por ejemplo, el Agente 4) escriben los logs en HDFS. El agent de cada nodo de recopilación puede seleccionar múltiples nodos de agregación para el balanceo de carga.

Figura 6-24 Flume en cascada



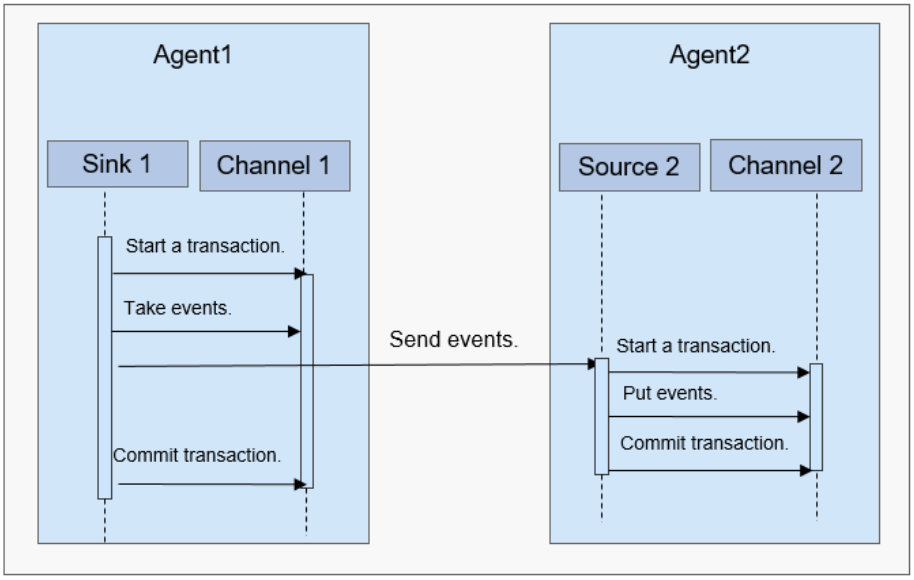
Para obtener más información sobre la arquitectura y los principios de Flume, consulte **https://flume.apache.org/releases/1.9.0.html**.

Principio

Confiabilidad entre agents

Figura 6-25 muestra el intercambio de datos entre agents.

Figura 6-25 Proceso de transmisión de datos



1. Flume garantiza una transmisión de datos confiable basada en transacciones. Cuando los datos fluyen de un agente a otro agente, las dos transacciones tienen efecto. El sink del Agent 1 (agente que envía un mensaje) necesita obtener un mensaje de un channel y envía el mensaje al Agent 2 (agente que recibe el mensaje). Si Agent 2 recibe y procesa con éxito el mensaje, Agent 1 enviará una transacción, indicando una transmisión de datos satisfactoria y confiable.
2. Cuando Agent 2 recibe el mensaje enviado por el Agent 1 e inicia una nueva transacción, después de que los datos se procesan con éxito (escritos en un channel), Agent 2 envía la transacción y envía una respuesta satisfactoria al Agent 1.
3. Antes de una operación de confirmación, si la transmisión de datos falla, la última transcripción comienza y retransmite los datos que no se transmiten la última vez. La operación de confirmación ha escrito la transacción en un disco. Por lo tanto, la última transacción puede continuar después de que el proceso falle y se restaure.

6.7.2 Relación entre Flume y otros componentes

Relación entre Flume y HDFS

Si HDFS está configurado como el sink de Flume, HDFS funciona como el sistema de almacenamiento de datos final de Flume. Flume instala, configura y escribe todos los datos transmitidos en HDFS.

Para obtener más información, consulte **Escenario típico: recopilar registros estáticos locales y cargarlos en HDFS** y **Escenario típico: recopilar registros dinámicos locales y cargarlos en HDFS**.

Relación entre Flume y HBase

Si HBase se configura como el sink de Flume, HBase funciona como el sistema de almacenamiento de datos final de Flume. Flume escribe todos los datos transmitidos en HBase basándose en configuraciones. Para obtener más información, consulte **Escenario típico: recopilar registros estáticos locales y cargarlos en HBase**.

6.7.3 Funciones de código abierto mejoradas de Flume

Funciones de código abierto mejoradas de Flume

- Mejora de la velocidad de transmisión: Se pueden especificar múltiples líneas en lugar de solo una línea de datos como un evento. Esto mejora la eficiencia de la ejecución de código y reduce los tiempos de escritura en disco.
- Transferencia de archivos binarios ultra grandes: De acuerdo con el uso actual de la memoria, Flume ajusta automáticamente la memoria utilizada para transferir archivos binarios ultra grandes para evitar la falta de memoria.
- Apoyo a la personalización de los preparativos antes y después de la transmisión: Flume admite scripts personalizados que se ejecutan antes o después de la transmisión para hacer los preparativos.
- Gestión de alarmas de clientes: Flume recibe alarmas de clientes de Flume a través de MonitorServer e informa de las alarmas al centro de gestión de alarmas en MRS Manager.

6.8 HBase

6.8.1 Principios básicos de HBase

HBase lleva a cabo el almacenamiento de datos. HBase es un sistema de almacenamiento distribuido de código abierto, orientado a columnas que es adecuado para almacenar cantidades masivas de datos no estructurados o semiestructurados. Cuenta con alta confiabilidad, alto rendimiento y escalabilidad flexible, y admite lectura/escritura de datos en tiempo real. Para obtener más información acerca de HBase, consulte **https://hbase.apache.org/**.

Las características típicas de una tabla almacenada en HBase son las siguientes:

- Tabla grande (BigTable): Una tabla contiene cientos de millones de filas y millones de columnas.
- Orientado a columnas: almacenamiento orientado a columnas, recuperación y control de permisos
- Disperso: las columnas nulas de la tabla no ocupan ningún espacio de almacenamiento.

Además, MRS HBase admite indexación secundaria para permitir que se creen índices para valores de columna de modo que los datos se puedan filtrar por columna usando API HBase nativas.

Arquitectura de HBase

Un clúster de HBase consta de procesos de HMaster activos y en espera y múltiples procesos de RegionServer.

Figura 6-26 Arquitectura de HBase

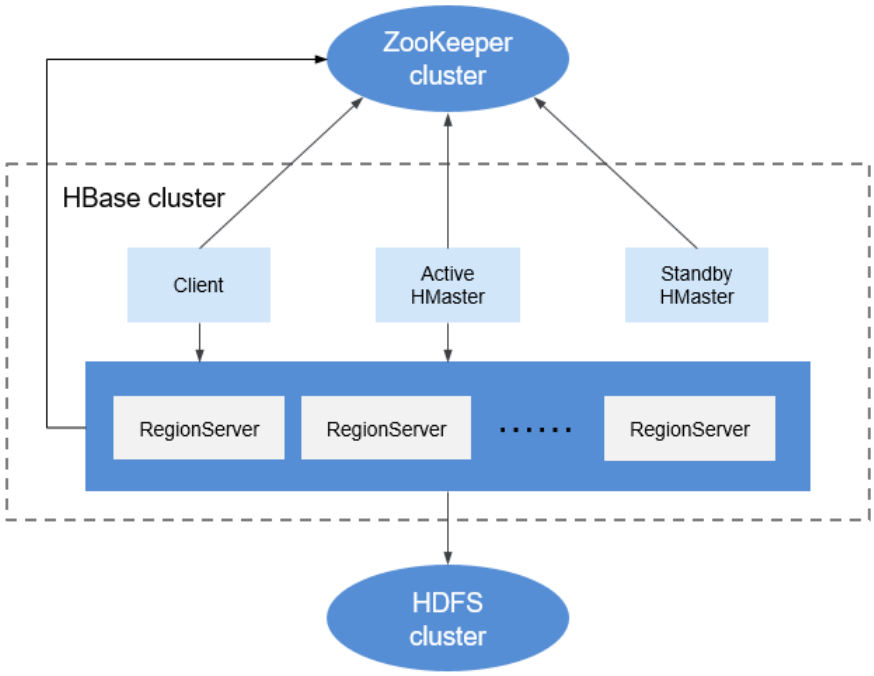


Tabla 6-6 Descripción del módulo

Módulo	Descripción
Master	Master también se llama HMaster. En modo de HA, HMaster consiste en un HMaster activo y un HMaster en espera. ● Active Master: gestiona RegionServer en HBase, incluida la creación, eliminación, modificación y consulta de una tabla, equilibra la carga de RegionServer, ajusta la distribución de Region, divide Region y distribuye Region después de dividirse, y migra Region después de que RegionServer caduque. ● Master en espera: se hace cargo de los servicios cuando el HMaster activo está defectuoso. El HMaster activo original se degrada al HMaster en espera después de rectificar la falla.
Client	Client se comunica con Master para la gestión y con RegionServer para la protección de datos mediante el mecanismo de llamada a procedimiento remoto (RPC) de HBase.
RegionServer	RegionServer proporciona servicios de lectura y escritura de datos de tablas como una unidad de procesamiento de datos y computación en HBase. RegionServer se implementa con DataNodes de clústeres HDFS para almacenar datos.
ZooKeeper cluster	ZooKeeper ofrece servicios de coordinación distribuida para procesos en clústeres de HBase. Cada RegionServer está registrado en ZooKeeper para que Master activo pueda obtener el estado de salud de cada RegionServer.

Módulo	Descripción
HDFS cluster	HDFS proporciona servicios de almacenamiento de archivos altamente confiables para HBase. Todos los datos de HBase se almacenan en el HDFS.

Principios de HBase

- **Modelo de datos de HBase**

  HBase almacena los datos en tablas, como se muestra en **Figura 6-27**. Los datos de una tabla se dividen en varias Regiones, que el Master asigna a RegionServers para su gestión.

  Cada Region contiene datos dentro de un rango de RowKey. Una tabla de datos de HBase contiene solo un Region al principio. A medida que aumenta el número de datos y alcanza el límite superior de la capacidad de Region, el Region se divide en dos Regions. Puede definir el rango RowKey de un Region al crear una tabla o definir el tamaño de Region en el archivo de configuración.

Figura 6-27 Modelo de datos de HBase

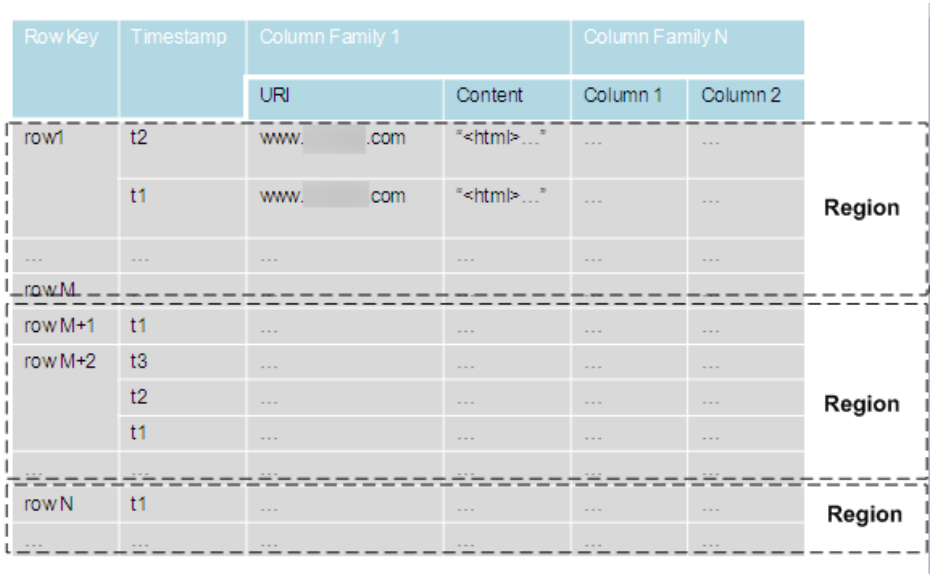


Tabla 6-7 Conceptos

Módulo	Descripción
RowKey	Similar a la clave principal de una tabla de relaciones, que es el ID único de los datos de cada fila. Un RowKey puede ser una cadena, un entero o una cadena binaria. Todos los registros se almacenan después de ser ordenados por RowKey.
Timestamp	La marca de tiempo de una operación de datos. Los datos se pueden especificar con diferentes versiones por sello de tiempo. Los datos de diferentes versiones en cada celda se almacenan por tiempo en orden descendente.

Módulo	Descripción
Cell	Unidad de almacenamiento mínima de HBase, que consta de claves y valores. Una clave consta de seis campos, a saber, row, column family, column qualifier, timestamp, type, and MVCC version. Los valores son los objetos de datos binarios.
Column Family	Una o varias familias de columnas horizontales forman una tabla. Una familia de columnas puede consistir en varias columnas aleatorias. Una columna es una etiqueta bajo una familia de columnas, que se puede agregar según sea necesario cuando se escriben datos. La familia de columnas admite la expansión dinámica, por lo que no es necesario que el número y el tipo de columnas estén predefinidos. Las columnas de una tabla en HBase están escasamente distribuidas. El número y el tipo de columnas en diferentes filas pueden ser diferentes. Cada familia de columnas tiene el tiempo de vida independiente (TTL). Solo puede bloquear la fila. Las operaciones de la fila de una familia de columnas son las mismas que las de las demás filas.
Column	Similar a las bases de datos tradicionales, las tablas de HBase también utilizan columnas para almacenar datos del mismo tipo.

- **Almacenamiento de datos de RegionServer**

  RegionServer gestiona las regiones asignadas por HMaster. **Figura 6-28** muestra la estructura de almacenamiento de datos de RegionServer.

  Figura 6-28 Estructura de almacenamiento de datos de RegionServer

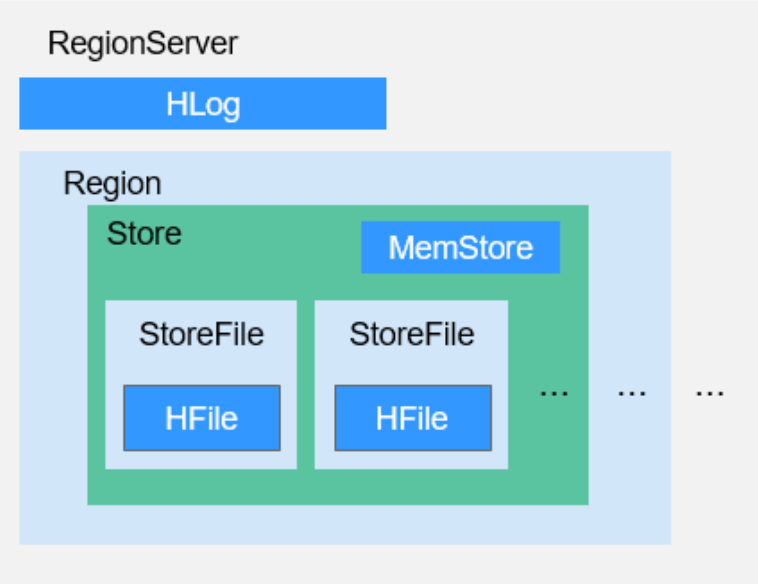


Tabla 6-8 enumera cada componente de la región descrito en **Figura 6-28**.

Tabla 6-8 Descripción de la estructura de la región

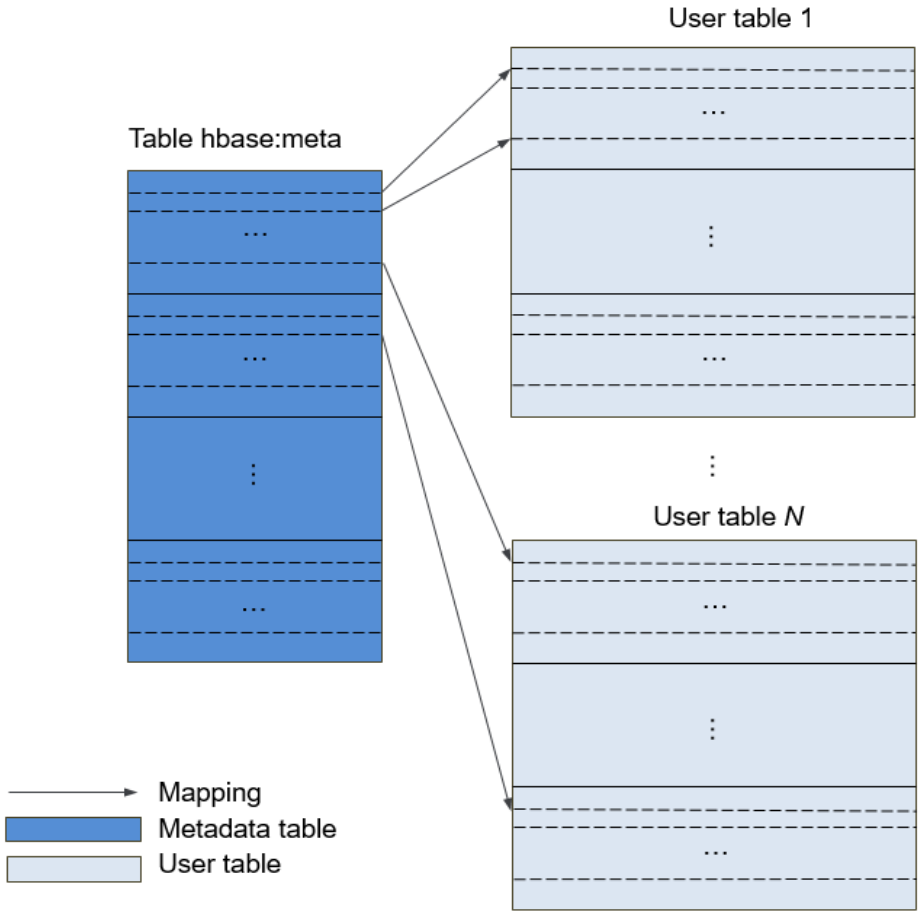
Módulo	Descripción
Store	Un Region se compone de una o varias Stores. Cada Store asigna una familia de columnas en **Figura 6-27**.
MemStore	Un Store contiene un MemStore. El MemStore almacena en caché los datos insertados en un Region por el cliente. Cuando la capacidad de MemStore alcanza el límite superior, RegionServer vacia los datos de MemStore a HDFS.
StoreFile	Los datos enviados al HDFS se almacenan como StoreFile en el HDFS. A medida que se insertan más datos, se generan múltiples StoreFiles en un Store. Cuando el número de StoreFiles alcanza el límite superior, RegionServer fusiona varios StoreFiles en un StoreFile grande.
HFile	HFile define el formato de almacenamiento de StoreFiles en un sistema de archivos. HFile es la implementación subyacente de StoreFile.
HLog	HLogs evitan la pérdida de datos cuando RegionServer es defectuoso. Múltiples regiones en un RegionServer comparten el mismo HLog.

- **Tabla de metadatos**

  La tabla de metadatos es una tabla HBase especial, que es utilizada por el cliente para localizar una región. La tabla de metadatos incluye una tabla **hbase:meta** para registrar información de región de las tablas de usuario, como la ubicación de región y las RowKey de inicio y fin.

  Figura 6-29 muestra la relación de asignación entre las tablas de metadatos y las tablas de usuario.

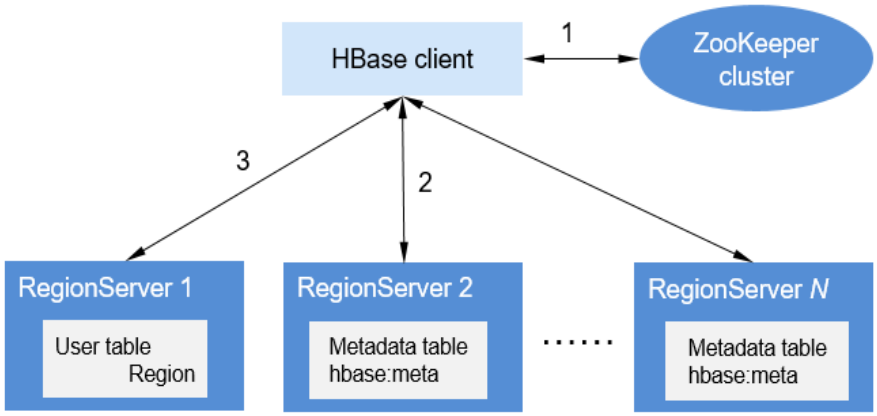
Figura 6-29 Asignación de relaciones entre tablas de metadatos y tablas de usuario



- **Proceso de operación de datos**

  Figura 6-30 muestra el proceso de operación de datos de HBase.

Figura 6-30 Procesamiento de datos



a. Cuando agrega, elimina, modifica y consulta datos de HBase, el cliente de HBase primero se conecta a ZooKeeper para obtener información sobre el RegionServer donde se encuentra la tabla **hbase:meta**. Si modifica el espacio de nombres, como

crear y eliminar una tabla, debe acceder a HMaster para actualizar la información de metadatos.

b. HBase Client se conecta al RegionServer donde se encuentra la región de la tabla **hbase:meta** y obtiene la ubicación RegionServer donde reside la región de la tabla de usuario.

c. A continuación, el HBase client se conecta al RegionServer donde se encuentra la región de la tabla de usuario y emite un comando de operación de datos al RegionServer. El RegionServer ejecuta el comando.

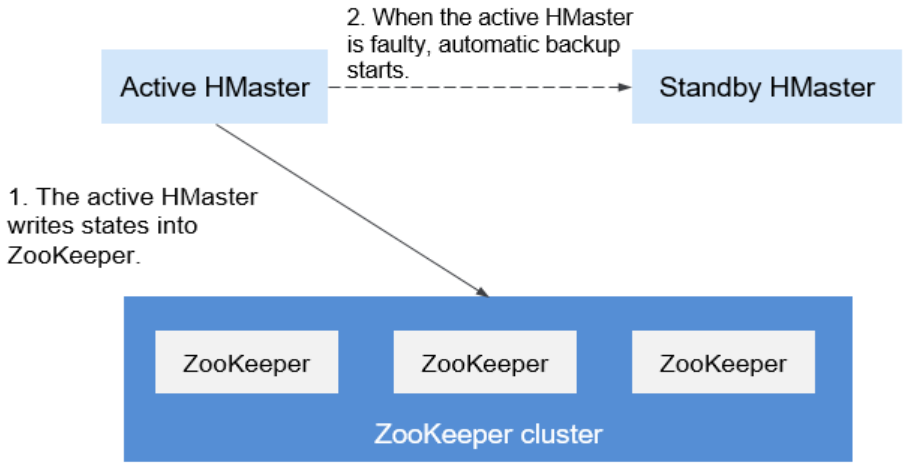
Para mejorar la eficiencia del procesamiento de datos, el cliente HBase almacena en caché la información de región de la tabla **hbase:meta** y la tabla de usuario. Cuando una aplicación inicia una segunda operación de datos, el HBase client consulta la información de región de la memoria. Si no se encuentra ninguna coincidencia en la memoria, el HBase client realiza las operaciones anteriores para obtener información de región.

6.8.2 Solución de HBase HA

HBase HA

HMaster en HBase asigna Regions. Cuando se detiene un servicio de RegionServer, HMaster migra la región correspondiente a otro RegionServer. La función HMaster HA se incorpora para evitar que las funciones HBase se vean afectadas por el punto único de falla (SPOF) de HMaster.

Figura 6-31 Arquitectura de implementación de HMaster HA



La arquitectura HA de HMaster se implementa creando el nodo de ZooKeeper efímero en un clúster de ZooKeeper.

Al iniciar, los nodos de HMaster intentan crear un znode maestro en el clúster de ZooKeeper. El nodo HMaster que crea el znode maestro primero se convierte en el HMaster activo, y el otro es el HMaster en espera.

Agregará eventos de vigilancia al nodo maestro. Si se detiene el servicio en el HMaster activo, el HMaster activo se desconecta del clúster ZooKeeper. Después de que la sesión expira, el HMaster activo desaparece. El HMaster en espera detecta la desaparición del

HMaster activo a través de eventos de vigilancia y crea un nodo de master para convertirse en el activo. A continuación, se completa la conmutación activa/en espera. Si el nodo fallido detecta la existencia del nodo de master después de reiniciarse, entra en el estado de standby y agrega eventos de vigilancia al nodo de master.

Cuando el cliente accede a la HBase, primero obtiene la dirección del HMaster basándose en la información del nodo de master en el ZooKeeper y luego establece una conexión con el HMaster activo.

6.8.3 Relación con otros componentes

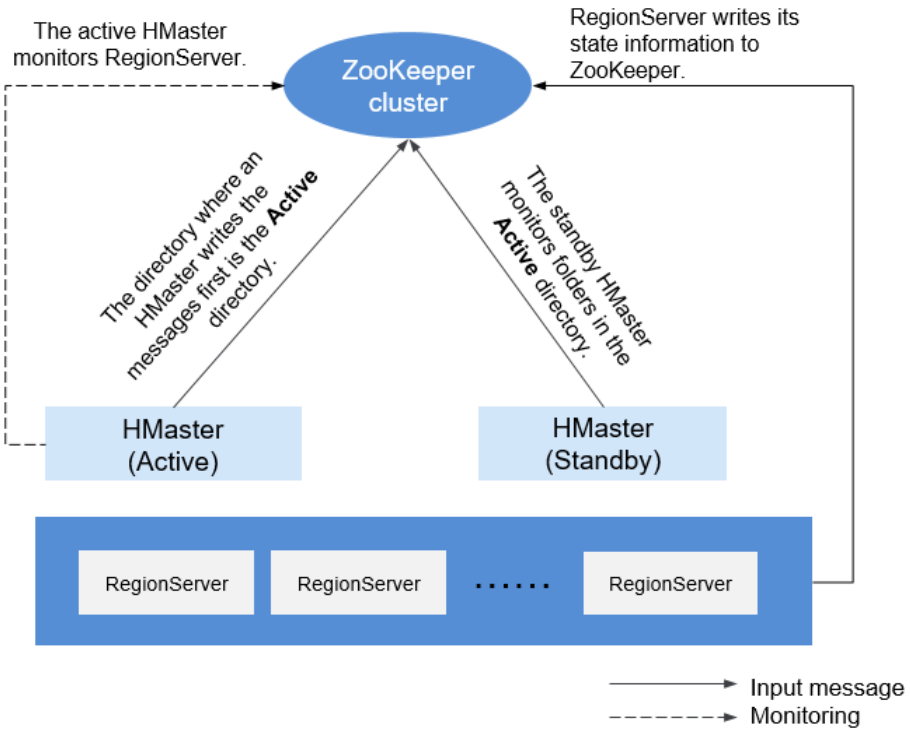
Relación entre HDFS y HBase

HDFS es el subproyecto de Apache Hadoop. HBase utiliza el sistema de archivos distribuido de Hadoop (HDFS) como sistema de almacenamiento de archivos. HBase se encuentra en la capa de almacenamiento estructurado. El HDFS proporciona soporte altamente confiable para almacenamiento de capa inferior de HBase. Todos los archivos de datos de HBase se pueden almacenar en el HDFS, excepto algunos archivos de registro generados por HBase.

Relación entre ZooKeeper y HBase

Figura 6-32 describe la relación entre ZooKeeper y HBase.

Figura 6-32 Relación entre ZooKeeper y HBase



1. HRegionServer se registra en ZooKeeper en el nodo Efímero. ZooKeeper almacena la información de HBase, incluidos los metadatos de HBase y las direcciones HMaster.
2. HMaster detecta el estado de salud de cada HRegionServer usando ZooKeeper y los monitorea.

3. HBase puede desplegar varios HMasters (como HDFS NameNode). Cuando el nodo HMatser activo está defectuoso, el nodo HMaster en espera obtiene la información de estado de todo el clúster usando ZooKeeper, lo que significa que fallas de punto único de HBase se pueden evitar usando ZooKeeper.

6.8.4 Funciones de código abierto mejoradas de HBase

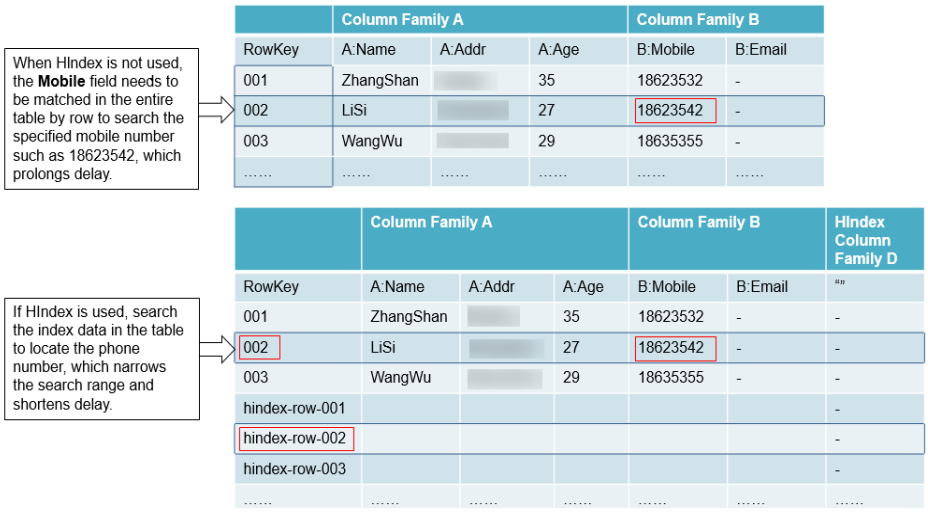
HIndex

HBase es una base de datos de almacenamiento distribuido del tipo Key-Value. Los datos de una tabla se ordenan en el orden alfabético basado en las claves de fila. Si consulta datos basados en una clave de fila especificada o analiza datos en la escala de una clave de fila especificada, HBase puede localizar rápidamente los datos de destino, lo que mejora la eficiencia.

Sin embargo, en la mayoría de los escenarios reales, debe consultar los datos cuyo valor de columna sea *XXX*. HBase proporciona la función Filter para consultar datos con un valor de columna específico. Todos los datos se analizan en el orden de las claves de fila y, a continuación, los datos se comparan con el valor de columna específico hasta que se encuentran los datos requeridos. La función Filter analiza algunos datos innecesarios para obtener los únicos datos necesarios. Por lo tanto, la función Filter no puede cumplir con los requisitos de consultas frecuentes con altos estándares de rendimiento.

HBase HIndex está diseñado para abordar estos problemas. HBase HIndex permite a HBase consultar datos basados en valores de columna específicos.

Figura 6-33 HIndex



- No se admite la actualización continua para los datos de índice.
- Restricciones de los índices combinados:
  - Todas las columnas implicadas en índices combinados deben introducirse o eliminarse en una sola mutación. De lo contrario, se producirá una incoherencia.

    Índice: **IDX1=>cf1:[q1->datatype],[q2];cf2:[q2->datatype]**

    Operaciones de escritura correctas:

    ```
    Put put = new Put(Bytes.toBytes("row"));
    put.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q1"),
    ```



```
Bytes.toBytes("valueA"));
put.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q2"),
Bytes.toBytes("valueB"));
put.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q2"),
Bytes.toBytes("valueC"));
table.put(put);
```

Operaciones de escritura incorrectas:

```
Put put1 = new Put(Bytes.toBytes("row"));
put1.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q1"),
Bytes.toBytes("valueA"));
table.put(put1);
Put put2 = new Put(Bytes.toBytes("row"));
put2.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q2"),
Bytes.toBytes("valueB"));
table.put(put2);
Put put3 = new Put(Bytes.toBytes("row"));
put3.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q2"),
Bytes.toBytes("valueC"));
table.put(put3);
```

- La consulta basada en condiciones combinadas sólo se admite cuando la columna de índice combinada contiene criterios de filtro, o no se especifican StartRow y StopRow para algunas columnas de índice.

  Índice: **IDX1=>cf1:[q1->datatype],[q2];cf2:[q1->datatype]**

  Operaciones de consulta correctas:

```
scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',>=,'binary:valueA',true,true
) AND SingleColumnValueFilter('cf1','q2',>=,'binary:valueB',true,true)
AND SingleColumnValueFilter('cf2','q1',>=,'binary:valueC',true,true) "}

scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',=,'binary:valueA',true,true)
 AND SingleColumnValueFilter('cf1','q2',>=,'binary:valueB',true,true)" }

scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',>=,'binary:valueA',true,true
) AND SingleColumnValueFilter('cf1','q2',>=,'binary:valueB',true,true)
AND
SingleColumnValueFilter('cf2','q1',>=,'binary:valueC',true,true)",STARTRO
W=>'row001',STOPROW=>'row100'}
```

  Operaciones de consulta incorrectas:

```
scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',>=,'binary:valueA',true,true
) AND SingleColumnValueFilter('cf1','q2',>=,'binary:valueB',true,true)
AND SingleColumnValueFilter('cf2','q1',>=,'binary:valueC',true,true)
AND SingleColumnValueFilter('cf2','q2',>=,'binary:valueD',true,true)"}

scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',=,'binary:valueA',true,true)
 AND SingleColumnValueFilter('cf2','q1',>=,'binary:valueC',true,true)" }

scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',=,'binary:valueA',true,true)
 AND SingleColumnValueFilter('cf2','q2',>=,'binary:valueD',true,true)" }

scan 'table',
{FILTER=>"SingleColumnValueFilter('cf1','q1',=,'binary:valueA',true,true)
 AND
SingleColumnValueFilter('cf1','q2',>=,'binary:valueB',true,true)" ,STARTR
OW=>'row001',STOPROW=>'row100' }
```

- No configure explícitamente ninguna política de división para tablas con datos de índice.
- Otras operaciones de mutación, tales como **increment** y **append** no son compatibles.
- No se admite el índice de la columna con **maxVersions** mayor que 1.

- La columna de índice de datos de una fila no se puede actualizar.

  Índice 1: **IDX1=>cf1:[q1->datatype],[q2];cf2:[q1->datatype]**

  Índice 2: **IDX2=>cf2:[q2->datatype]**

  Operaciones de actualización correctas:

```
Put put1 = new Put(Bytes.toBytes("row"));
put1.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q1"),
Bytes.toBytes("valueA"));
put1.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q2"),
Bytes.toBytes("valueB"));
put1.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q1"),
Bytes.toBytes("valueC"));
put1.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q2"),
Bytes.toBytes("valueD"));
table.put(put1);

Put put2 = new Put(Bytes.toBytes("row"));
put2.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q3"),
Bytes.toBytes("valueE"));
put2.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q3"),
Bytes.toBytes("valueF"));
table.put(put2);
```

  Operaciones de actualización incorrectas:

```
Put put1 = new Put(Bytes.toBytes("row"));
put1.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q1"),
Bytes.toBytes("valueA"));
put1.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q2"),
Bytes.toBytes("valueB"));
put1.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q1"),
Bytes.toBytes("valueC"));
put1.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q2"),
Bytes.toBytes("valueD"));
table.put(put1);

Put put2 = new Put(Bytes.toBytes("row"));
put2.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q1"),
Bytes.toBytes("valueA_new"));
put2.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("q2"),
Bytes.toBytes("valueB_new"));
put2.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q1"),
Bytes.toBytes("valueC_new"));
put2.addColumn(Bytes.toBytes("cf2"), Bytes.toBytes("q2"),
Bytes.toBytes("valueD_new"));
table.put(put2);
```

- La tabla a la que se agrega un índice no puede contener un valor superior a 32 KB.
- Si los datos de usuario se eliminan debido a la expiración del TTL a nivel de columna, los datos de índice correspondientes no se eliminan inmediatamente. Se eliminará en la operación de compactación principal.
- El TTL de la familia de columnas de usuario no se puede modificar después de crear el índice.
  - Si el TTL de una familia de columnas aumenta después de crear un índice, elimine el índice y vuelva a crear uno. De lo contrario, algunos datos de índice generados se eliminarán antes de que se eliminen los datos del usuario.
  - Si el valor TTL de la familia de columnas disminuye después de crear un índice, los datos del índice se eliminarán después de eliminar los datos del usuario.
- La consulta de índice no admite la operación inversa y los resultados de la consulta están desordenados.
- El índice no admite la operación **clone snapshot**.

- La tabla de índice debe usar HIndexWALPlayer para reproducir registros. WALPlayer no se puede usar para reproducir registros.

```
hbase org.apache.hadoop.hbase.hindex.mapreduce.HIndexWALPlayer
Usage: WALPlayer [options] <wal inputdir> <tables> [<tableMappings>]
Read all WAL entries for <tables>.
If no tables ("") are specific, all tables are imported.
(Careful, even -ROOT- and hbase:meta entries will be imported in that case.)
Otherwise <tables> is a comma separated list of tables.

The WAL entries can be mapped to new set of tables via <tableMapping>.
<tableMapping> is a command separated list of targettables.
If specified, each table in <tables> must have a mapping.

By default WALPlayer will load data directly into HBase.
To generate HFiles for a bulk data load instead, pass the option:
  -Dwal.bulk.output=/path/for/output
  (Only one table can be specified, and no mapping is allowed!)
Other options: (specify time range to WAL edit to consider)
  -Dwal.start.time=[date|ms]
  -Dwal.end.time=[date|ms]
For performance also consider the following options:
  -Dmapreduce.map.speculative=false
  -Dmapreduce.reduce.speculative=false
```

- Cuando se ejecuta el comando **deleteall** para la tabla de índices, el rendimiento es bajo.
- La tabla de índice no admite HBCK. Para utilizar HBCK para reparar la tabla de índice, elimine primero los datos de índice.

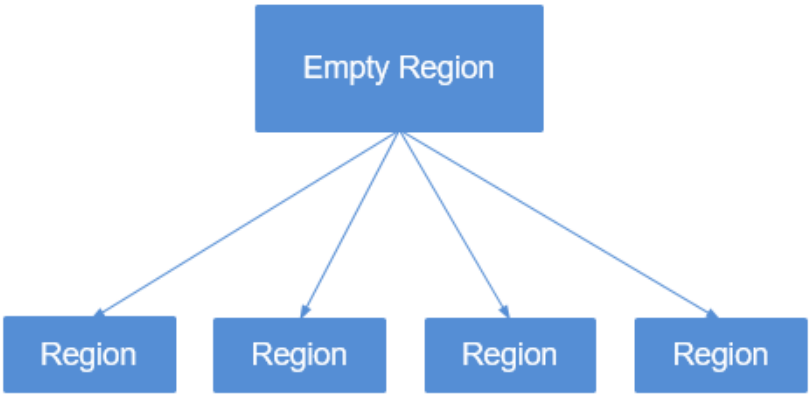
División multipunto

Cuando crea tablas que están divididas previamente por región en HBase, es posible que no conozca la tendencia de distribución de datos, por lo que la división por región puede ser inapropiada. Después de que el sistema funcione durante un período, las regiones deben dividirse de nuevo para lograr un mejor rendimiento. Solo se pueden dividir las regiones vacías.

La función de división de regiones proporcionada con HBase divide regiones solo cuando alcanzan el umbral. Esto se llama "división de un solo punto".

Para lograr un mejor rendimiento cuando las regiones se dividen en función de los requisitos del usuario, se desarrolla la división multipunto, que también se denomina "división dinámica". Es decir, una región vacía se divide previamente en múltiples regiones para evitar el deterioro del rendimiento causado por el espacio insuficiente de la región.

Figura 6-34 División multipunto



Limitación de conexión

Demasiadas sesiones significan que se están ejecutando demasiadas consultas y tareas de MapReduce en HBase, lo que compromete el rendimiento de HBase e incluso provoca el rechazo del servicio. Puede configurar parámetros para limitar el número máximo de sesiones que se pueden establecer entre el cliente y el servidor HBase para lograr la protección de sobrecarga HBase.

Recuperación ante desastres mejorada

Las capacidades de recuperación ante desastres (DR) entre los clústeres activo y en espera pueden mejorar el HA de los datos de HBase. El clúster activo proporciona servicios de datos y el clúster en espera realiza copias de seguridad de los datos. Si el clúster activo es defectuoso, el clúster en espera se hace cargo de los servicios de datos. En comparación con la función de replicación de código abierto, esta función se mejora de la siguiente manera:

1. La función de lista blanca de clústeres en espera solo es aplicable a enviar datos a una dirección IP de clúster especificada.
2. En la versión de código abierto, la replicación se sincroniza en función de WAL, y la copia de respaldo de datos se implementa reproduciendo WAL en el clúster en espera. Para las operaciones de BulkLoad, dado que no se genera WAL, los datos no se replicarán en el clúster en espera. Al registrar las operaciones de BulkLoad en el WAL y sincronizarlas con el clúster en espera, el clúster en espera puede leer los registros de operación de BulkLoad a través de WAL y cargar HFile en el clúster activo al clúster en espera para implementar la copia de respaldo de datos.
3. En la versión de código abierto, HBase filtra las ACL. Por lo tanto, la información de ACL no se sincronizará con el clúster en espera. Al agregar un filtro (**org.apache.hadoop.hbase.replication.SystemTableWALEntryFilterAllowACL**), la información de ACL se puede sincronizar con el clúster en espera. Puede configurar **hbase.replication.filter.sytemWALEntryFilter** para habilitar el filtro e implementar la sincronización de ACL.
4. En cuanto a la restricción de solo lectura del clúster en espera, solo los superusuarios dentro del clúster en espera pueden modificar el HBase del clúster en espera. En otras palabras, los clientes HBase fuera del clúster en espera solo pueden leer el HBase del clúster en espera.

HBase MOB

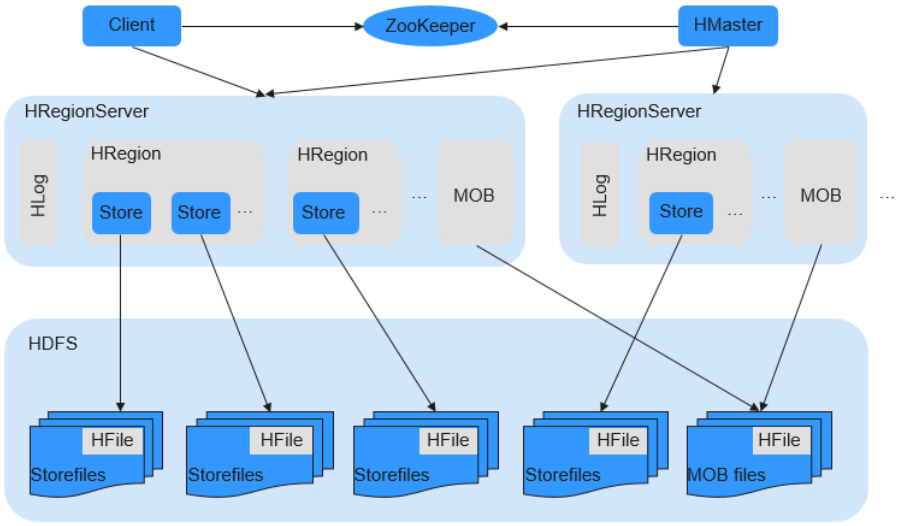
En los escenarios de aplicación reales, es necesario almacenar datos en varios tamaños, por ejemplo, datos de imagen y documentos. Los datos cuyo tamaño sea inferior a 10 MB se pueden almacenar en HBase. HBase puede ofrecer el mejor rendimiento de lectura y escritura para datos cuyo tamaño sea inferior a 100 KB. Si el tamaño de los datos almacenados en HBase es superior a 100 KB o incluso alcanza 10 MB y se inserta el mismo número de archivos de datos, la cantidad total de datos es grande, lo que provoca compactación y división frecuentes, alto consumo de CPU, alta frecuencia de E/S de disco y bajo rendimiento.

Los datos MOB (cuyo tamaño varía de 100 KB a 10 MB) se almacenan en un sistema de archivos (por ejemplo, HDFS) en formato HFile. Las herramientas expiredMobFileCleaner y Sweeper se utilizan para gestionar HFiles y guardar la dirección y la información de tamaño sobre los HFiles en el almacén de HBase como valores. Esto reduce en gran medida la compactación y la frecuencia de división en HBase y mejora el rendimiento.

Como se muestra en **Figura 6-35**, MOB indica mobstore almacenado en HRegion. Mobstore almacena claves y valores. En el que, una clave es la clave correspondiente en HBase, y un

valor es la dirección de referencia y el desplazamiento de datos almacenados en el sistema de archivos. Al leer datos, mobstore utiliza su propio escáner para leer objetos de datos clave-valor y utiliza la información de dirección y tamaño de datos en el valor para obtener datos de destino del sistema de archivos.

Figura 6-35 Principio de almacenamiento de datos MOB



HFS

El FileStream HBase (HFS) es un módulo de almacenamiento de archivos HBase independiente. Se utiliza en aplicaciones de capa superior MRS mediante la encapsulación de interfaces HBase y HDFS para proporcionar estas aplicaciones de capa superior con funciones tales como almacenamiento de archivos, lectura y eliminación.

En el ecosistema de Hadoop, HDFS y HBase enfrentan problemas difíciles en el almacenamiento masivo de archivos en algunos escenarios:

- Si se almacena un gran número de archivos pequeños en HDFS, el NameNode estará bajo gran presión.
- Algunos archivos grandes no se pueden almacenar directamente en HBase debido a las API de HBase y los mecanismos internos.

HFS está desarrollado para el almacenamiento mixto de archivos pequeños masivos y algunos archivos grandes en Hadoop. En pocas palabras, los archivos grandes pequeños (más pequeños que 10 MB) y algunos archivos grandes (más de 10 MB) deben almacenarse en tablas HBase.

Para tal escenario, HFS proporciona API de operación unificada similar a las API de función de HBase.

Múltiples RegionServers implementados en el mismo servidor

Se pueden implementar múltiples RegionServers en un nodo para mejorar la utilización de recursos de HBase.

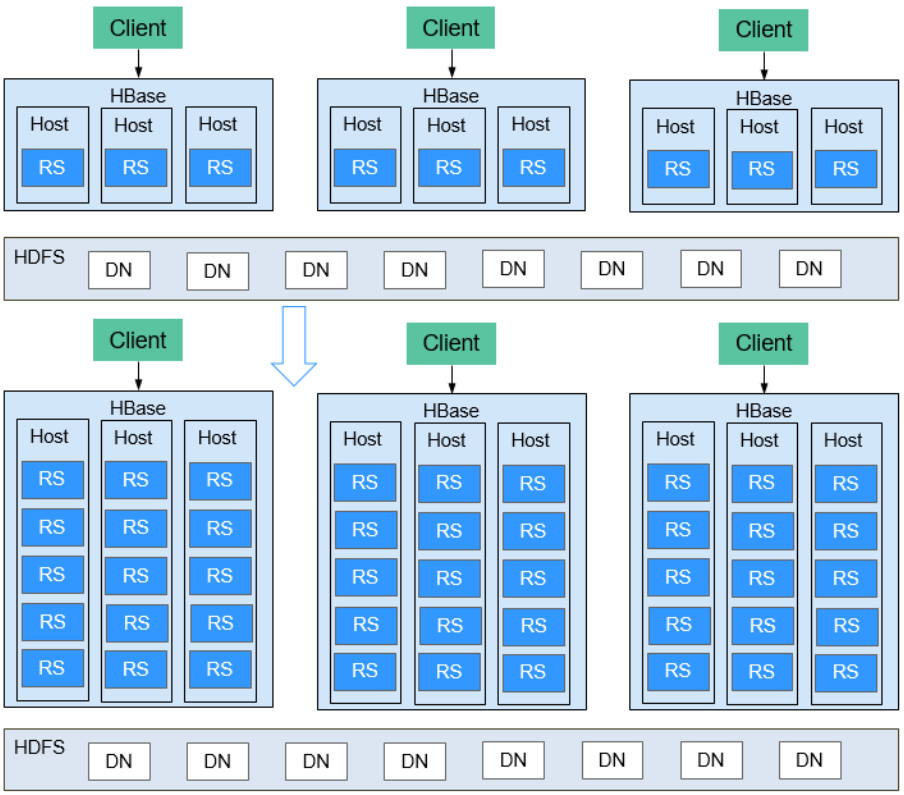
Si solo se despliega un RegionServer, la utilización de recursos es baja debido a las siguientes razones:

1. Un RegionServer admite un número limitado de regiones y, por lo tanto, los recursos de memoria y CPU no se pueden usar completamente.
2. Un solo RegionServer admite un máximo de 20 TB de datos, de los cuales dos copias requieren 40 TB y tres copias requieren 60 TB. En este caso, la capacidad de 96 TB no se puede agotar.
3. Rendimiento de escritura deficiente: se despliega un RegionServer en un servidor físico y solo existe un HLog. Solo se pueden escribir tres discos al mismo tiempo.

La utilización de recursos de HBase se puede mejorar cuando se despliegan múltiples RegionServers en el mismo servidor.

1. Un servidor físico se puede configurar con un máximo de cinco RegionServers. El número de RegionServers desplegados en cada servidor físico se puede configurar según sea necesario.
2. Se pueden utilizar recursos como memoria, discos y CPU.
3. Un servidor físico admite un máximo de cinco HLogs y permite que los datos se escriban en 15 discos al mismo tiempo, lo que mejora significativamente el rendimiento de escritura.

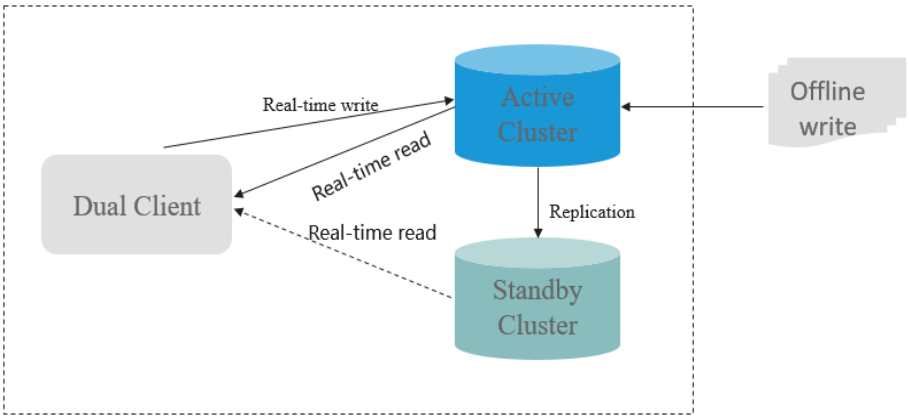
Figura 6-36 Utilización mejorada de los recursos de HBase



HBase de lectura dual

En el escenario de almacenamiento HBase, es difícil garantizar una estabilidad de consulta del 99.9% debido a GC, jitter de red y sectores defectuosos de discos. La función de lectura dual de HBase se agrega para cumplir con los requisitos de fallos bajos durante la lectura aleatoria de gran volumen de datos.

La función de lectura dual de HBase se basa en la capacidad de recuperación ante desastres de los clústeres activo y en espera. La probabilidad de que los dos grupos generen fallas al mismo tiempo es mucho menor que la de un grupo. El modo de acceso simultáneo de doble clúster se utiliza para garantizar la estabilidad de la consulta. Cuando un usuario inicia una solicitud de consulta, el servicio HBase de los dos clústeres se consulta al mismo tiempo. Si el clúster activo no devuelve ningún resultado después de un período de tiempo (el tiempo de fallo máximo tolerable), se pueden utilizar los datos del clúster con la respuesta más rápida. La siguiente figura muestra el principio de funcionamiento.



6.9 HDFS

6.9.1 Principios básicos de HDFS

Hadoop Distributed File System (HDFS) implementa lectura/escritura confiable y distribuida de cantidades masivas de datos. HDFS es aplicable al escenario en el que las características de lectura/escritura de datos "escriban una vez y lean varias veces". Sin embargo, la operación de escritura se realiza en secuencia, es decir, es una operación de escritura realizada durante la creación del archivo o una operación de adición realizada detrás del archivo existente. HDFS se asegura de que solo una persona que llama puede realizar la operación de escritura en un archivo, pero varias personas que llaman pueden realizar la operación de lectura en el archivo al mismo tiempo.

Arquitectura

El HDFS se compone de NameNodes activo y en espera y DataNodes múltiples, como se muestra en el **Figura 6-37**.

HDFS trabaja en arquitectura maestra/esclava. NameNodes se ejecuta en el nodo maestro (activo) y DataNodes se ejecuta en el nodo esclavo (en espera). ZKFC debería funcionar junto con el NameNodes.

La comunicación entre NameNodes y DataNodes se basa en el Protocolo de Control de Transmisión (TCP)/Protocolo de Internet (IP). Los NameNode, DataNode, ZKFC y JournalNode se pueden implementar en servidores de Linux.

Figura 6-37 Arquitectura HA HDFS

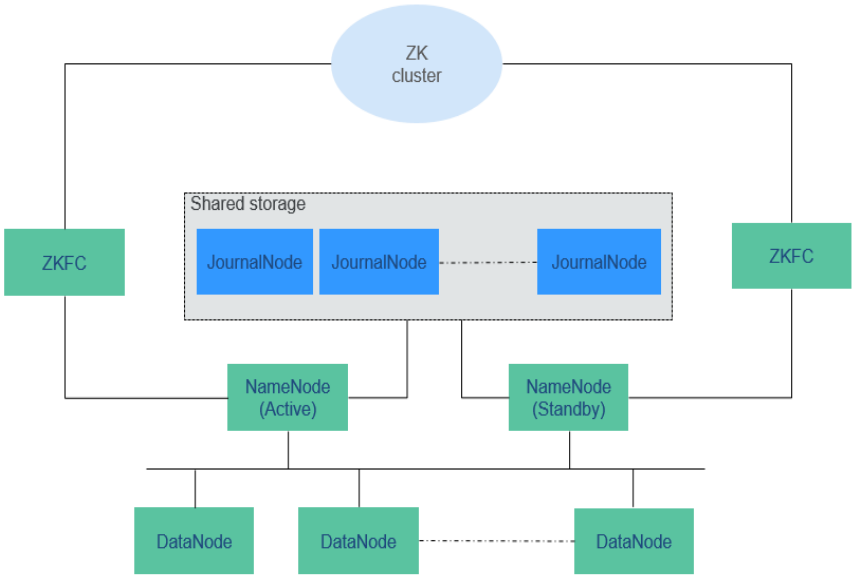


Tabla 6-9 describe las funciones de cada módulo mostrado en **Figura 6-37**.

Tabla 6-9 Descripción del módulo

Módulo	Descripción
NameNode	Un NameNode se utiliza para gestionar el espacio de nombres, la estructura de directorios y la información de metadatos de un sistema de archivos y proporcionar el mecanismo de copia de respaldo. El NameNode se clasifica en los dos tipos siguientes: ● NameNode activo: gestiona el espacio de nombres, mantiene la estructura de directorios y metadatos de los sistemas de archivos, y registra las relaciones de asignación entre bloques de datos y archivos a los que pertenecen los bloques de datos. ● Standby NameNode: se sincroniza con los datos en el NameNode activo y toma el control de los servicios del NameNode activo cuando el NameNode activo es defectuoso. ● Observer NameNode: se sincroniza con los datos del NameNode activo y procesa las solicitudes de lectura del cliente.
DataNode	Se utiliza un DataNode para almacenar bloques de datos de cada archivo e informar periódicamente al NameNode del estado de almacenamiento.
JournalNode	En el clúster de HA, sincroniza los metadatos entre los NameNodes activo y en espera.
ZKFC	Se debe implementar ZKFC para cada NameNode. Supervisa el estado de NameNode y escribe información de estado en ZooKeeper. ZKFC también tiene permisos para seleccionar el NameNode activo.
ZK Cluster	ZooKeeper es un servicio de coordinación que ayuda a la ZKFC a elegir el NameNode activo.

Módulo	Descripción
HttpFS gateway	HttpFS es un único proceso de gateway sin estado que proporciona la WebHDFS REST API para procesos externos y FileSystem API para HDFS. HttpFS se utiliza para la transmisión de datos entre diferentes versiones de Hadoop. También se utiliza como gateway para acceder al HDFS detrás de un firewall.

- **Arquitectura de HDFS HA**

  HA se utiliza para resolver el problema de SPOF de NameNode. Esta función proporciona un NameNode en espera para el NameNode activo. Cuando el NameNode activo está defectuoso, el NameNode en espera puede asumir rápidamente el control para proporcionar continuamente servicios para sistemas externos.

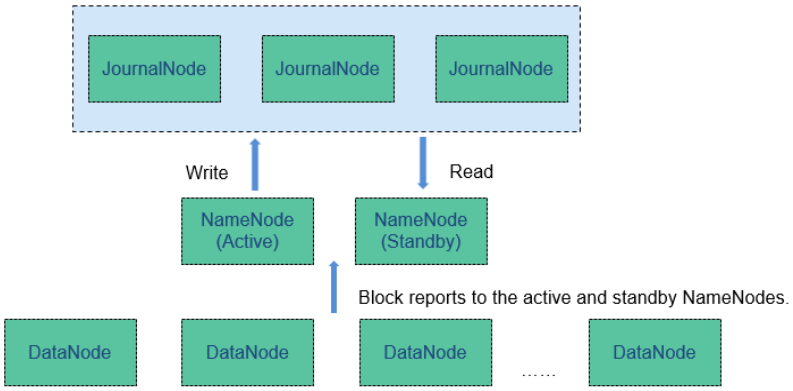
  En un escenario de HDFS HA típico, normalmente hay dos NameNodes. Uno está en el estado activo, y el otro en el estado de espera.

  Se requiere un sistema de almacenamiento compartido para admitir la sincronización de metadatos del NameNodes activo y en espera. Esta versión proporciona la solución de HA del Quorum Journal Manager (QJM), como se muestra en el documento **Figura 6-38**. Se utiliza un grupo de JournalNodes para sincronizar metadatos entre los NameNodes activo y en espera.

  Generalmente, se configura un número impar (2N+1) de JournalNodes y se requieren al menos tres JournalNodes. Para un mensaje de actualización de metadatos, la escritura de datos se considera exitosa siempre que la escritura de datos sea exitosa en JournalNodes N+1. En este caso, se permite una falla de escritura de datos de un máximo de N JournalNodes. Por ejemplo, cuando hay tres JournalNodes, se permite el error de escritura de datos de un JournalNode; cuando hay cinco JournalNodes, se permite el error de escritura de datos de dos JournalNodes.

  JournalNode es un proceso de daemon ligero y comparte un host con otros servicios de Hadoop. Se recomienda que el JournalNode se despliegue en el nodo de control para evitar el fallo de escritura de datos en el JournalNode durante la transmisión masiva de datos.

  Figura 6-38 Arquitectura de HDFS basada en QJM

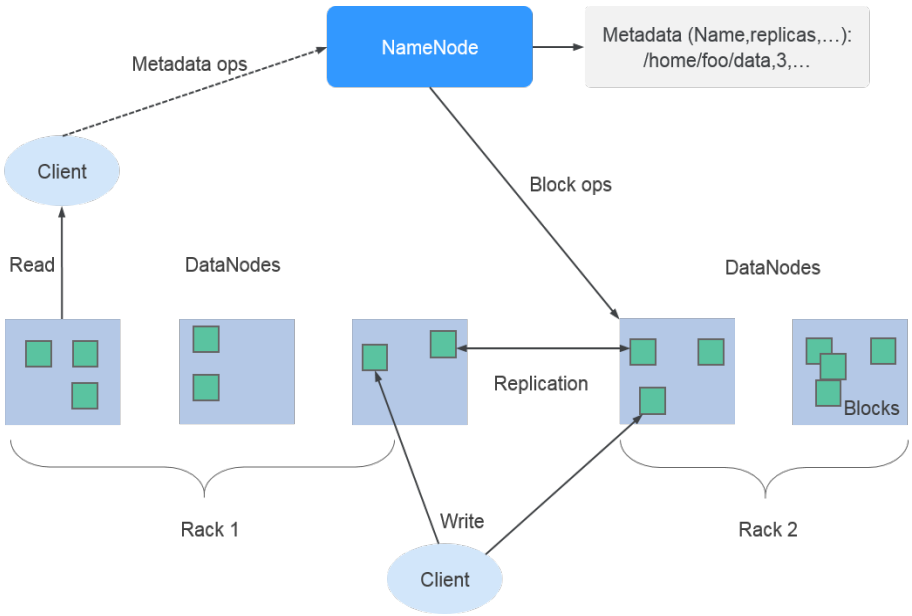


Principio

MRS utiliza el mecanismo de copia de HDFS para garantizar la confiabilidad de los datos. Un archivo de copia de respaldo se genera automáticamente por cada archivo guardado en HDFS, es decir, se generan dos copias en total. El número de copias HDFS se puede consultar mediante el parámetro **dfs.replication**.

- Cuando la especificación del nodo de Core del clúster MRS se establece en una unidad de disco duro no local (HDD) y el clúster solo tiene un nodo de Core, el número predeterminado de copias HDFS es 1. Si el número de nodos de Core en el clúster es mayor o igual a 2, el número predeterminado de copias HDFS es 2.
- Cuando la especificación del nodo Core del clúster MRS se establece en disco local y el clúster solo tiene un nodo de Core, el número predeterminado de copias HDFS es 1. Si hay dos nodos de Core en el clúster, el número predeterminado de copias HDFS es 2. Si el número de nodos de Core en el clúster es mayor o igual a 3, el número predeterminado de copias HDFS es 3.

Figura 6-39 Arquitectura de HDFS



El componente de HDFS de MRS admite las siguientes características:

- Soporta código de borrado, reduciendo la redundancia de datos al 50% y mejorando la confiabilidad. Además, se introduce la estructura de almacenamiento de bloques divididos para maximizar el uso de la capacidad de un único nodo y múltiples discos en un clúster existente. Después de introducir el proceso de codificación, se mejora el rendimiento de escritura de datos, y el rendimiento es cercano al de la redundancia multicopia.
- Admite la programación de nodos balanceados en HDFS y la programación de disco balanceado en un solo nodo, lo que mejora el rendimiento de almacenamiento de HDFS después del escalamiento horizontal del nodo o del disco.

Para obtener más información sobre la arquitectura y los principios de Hadoop, consulte **https://hadoop.apache.org/**.

6.9.2 Solución de HDFS HA

Fondo de HDFS HA

En versiones anteriores a Hadoop 2.0.0, SPOF se produce en el clúster HDFS. Cada clúster tiene un solo NameNode. Si el host donde se encuentra el NameNode es defectuoso, el clúster HDFS no se puede utilizar a menos que el NameNode se reinicie o se inicie en otro host. Esto afecta a la disponibilidad general de HDFS en los siguientes aspectos:

1. En el caso de un evento no planificado, como la avería del host, el clúster no estará disponible hasta que se reinicie el NameNode.
2. Las tareas de mantenimiento planificadas, como la actualización de software y hardware, harán que el clúster deje de funcionar.

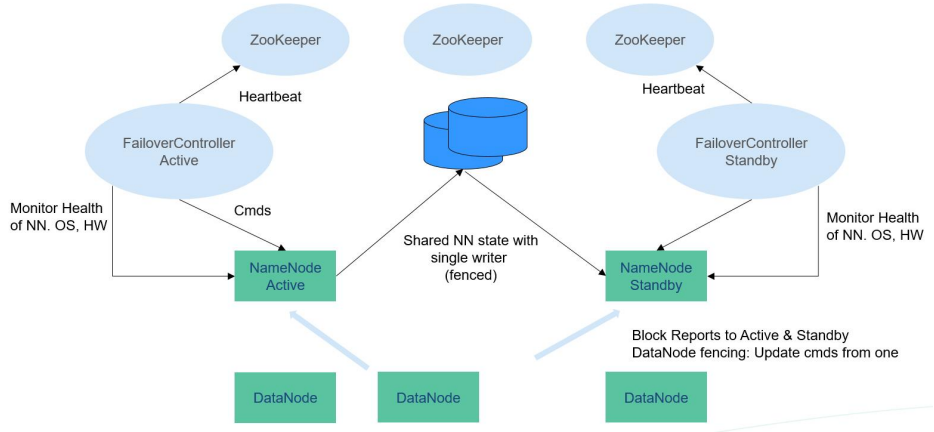
Para resolver los problemas anteriores, la solución HDFS HA permite una copia de respaldo de NameNode de intercambio en caliente para NameNodes en un clúster en modo automático o manual (configurable). Cuando un equipo falla (debido a un fallo de hardware), el NameNode activo/en espera cambia automáticamente en poco tiempo. Cuando es necesario mantener el NameNode activo, el administrador del clúster de MRS puede realizar manualmente una conmutación NameNode activa/en espera para garantizar la disponibilidad del clúster durante el mantenimiento.

Para obtener más información sobre la migración por falla automática de HDFS, consulte

http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html#Automatic_Failover

Implementación de HDFS HA

Figura 6-40 Implementación de HA típica



En un clúster de HA típico (como se muestra en **Figura 6-40**), dos NameNodes deben configurarse en dos servidores independientes, respectivamente. En cualquier punto de tiempo, un NameNode está en el estado activo, y el otro NameNode está en el estado de espera. El NameNode activo es responsable de todas las operaciones del cliente en el clúster, mientras que el NameNode en espera mantiene la sincronización con el nodo activo para proporcionar conmutación rápida si es necesario.

Para mantener los datos sincronizados entre sí, ambos nodos se comunican con un grupo de JournalNodes. Cuando el nodo activo modifica los metadatos de cualquier sistema de

archivos, almacenará el registro de modificación en la mayoría de estos JournalNodes. Por ejemplo, si hay tres JournalNodes, entonces el registro se guardará en dos de ellos al menos. El nodo en espera supervisa los cambios de JournalNodes y sincroniza los cambios del nodo activo. Basado en el registro de modificaciones, el nodo en espera aplica los cambios a los metadatos del sistema de archivos local. Una vez que se produce una conmutación, el nodo en espera puede garantizar que su estado es el mismo que el del nodo activo. Esto asegura que los metadatos del sistema de archivos están sincronizados entre los nodos activo y en espera si la conmutación se produce por la falla del nodo activo.

Para garantizar una conmutación rápida, el nodo en espera necesita tener la información de bloque más reciente. Por lo tanto, los DataNodes envían información de bloque y mensajes de latidos a dos NameNodes al mismo tiempo.

Es vital para un clúster de HA que solo uno de los NameNodes esté activo en cualquier momento. De lo contrario, el estado del espacio de nombres se dividiría en dos partes, arriesgando la pérdida de datos u otros resultados incorrectos. Para evitar el llamado "escenario de split-brain", el JournalNodes solo permitirá que un solo NameNode le escriba datos a la vez. Durante la conmutación, el NameNode que se activará asumirá el papel de escribir datos en JournalNodes. Esto evita eficazmente que el otro NameNodes esté en el estado activo, permitiendo que el nuevo nodo activo proceda con seguridad con la conmutación.

Para obtener más información acerca de la solución HDFS HA, visite el siguiente sitio Web:

http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

6.9.3 Relación entre HDFS y otros componentes

Relación entre HDFS y HBase

HDFS es un subproyecto de Apache Hadoop, que se utiliza como sistema de almacenamiento de archivos para HBase. HBase se encuentra en la capa de almacenamiento estructurado. HDFS proporciona soporte altamente confiable para almacenamiento de capa inferior de HBase. Todos los archivos de datos de HBase se pueden almacenar en el HDFS, excepto algunos archivos de registro generados por HBase.

Relación entre HDFS y MapReduce

- HDFS cuenta con alta tolerancia a fallas y alto rendimiento, y se puede implementar en hardware de bajo costo para almacenar datos de aplicaciones con conjuntos de datos masivos.
- MapReduce es un modelo de programación utilizado para el cálculo paralelo de grandes conjuntos de datos (mayores de 1 TB). Los datos computados por MapReduce provienen de múltiples orígenes de datos, como FileSystem locales, HDFS y bases de datos. La mayoría de los datos provienen del HDFS. El alto rendimiento de HDFS se puede utilizar para leer datos masivos. Después de ser computados, los datos se pueden almacenar en HDFS.

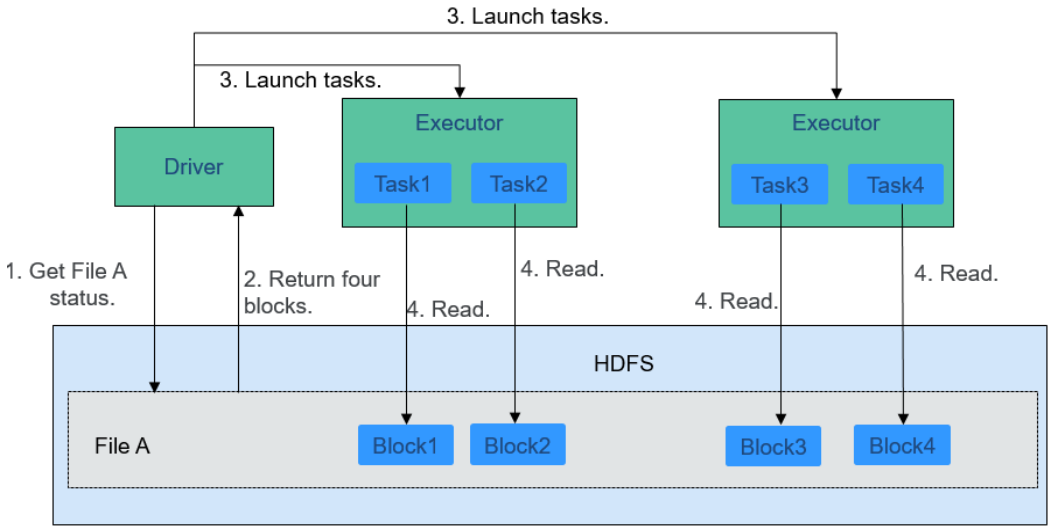
Relación entre HDFS y Spark

Los datos calculados por Spark provienen de múltiples fuentes de datos, como archivos locales y HDFS. La mayoría de los datos provienen de HDFS que pueden leer datos a gran escala para la computación paralela. Después de ser computados, los datos se pueden almacenar en HDFS.

Spark implica Driver y Executor. El Driver programa las tareas y el Executor ejecuta las tareas.

Figura 6-41 muestra cómo se leen los datos de un archivo.

Figura 6-41 Proceso de lectura de archivos

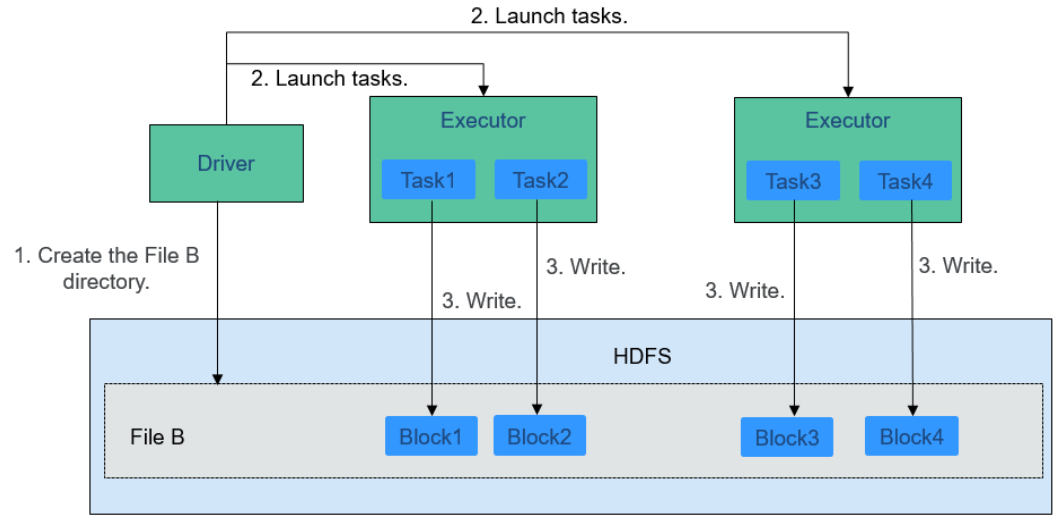


El proceso de lectura de archivos es el siguiente:

1. Driver se interconecta con HDFS para obtener la información del File A.
2. El HDFS devuelve la información de block detallada acerca de este archivo.
3. Driver establece un grado paralelo basado en la cantidad de datos de block y crea varias tasks para leer los blocks de este archivo.
4. Executor ejecuta las tareas y lee los blocks detallados como parte del conjunto de datos distribuido resistentes (RDD).

Figura 6-42 muestra cómo se escriben los datos en un archivo.

Figura 6-42 Proceso de escritura de archivos



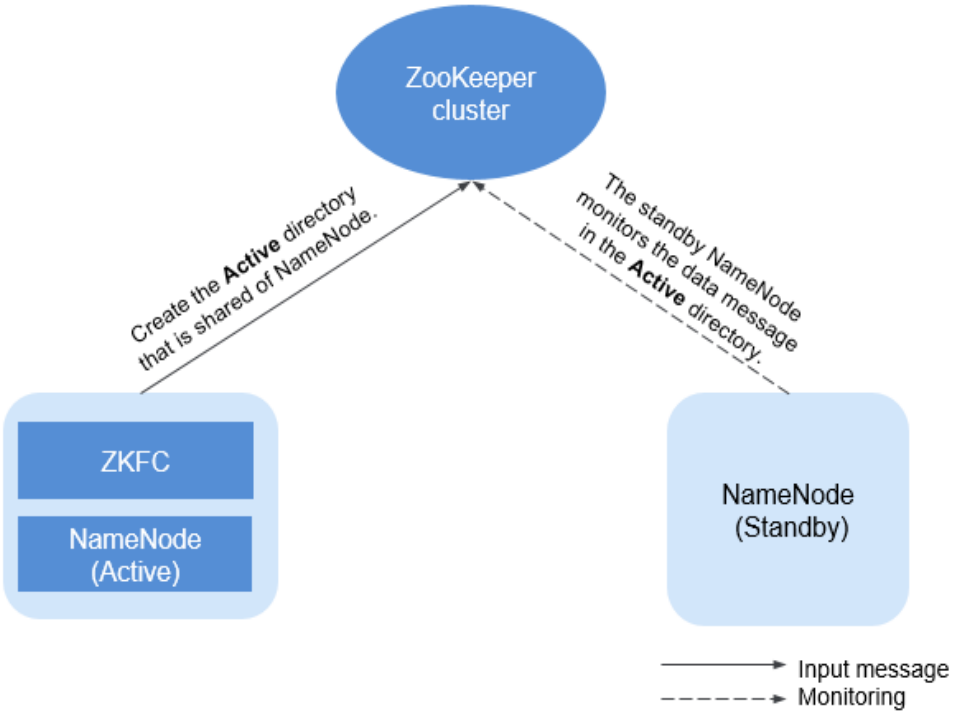
El proceso de escritura de archivos es el siguiente:

1. Driver crea un directorio donde se va a escribir el archivo.
2. Basándose en el estado de distribución de RDD, se calcula el número de tasks relacionadas con la escritura de datos, y estas tareas se envían al Executor.
3. Ejecutor ejecuta estas tareas y escribe los datos RDD calculados en el directorio creado por 1.

Relación entre HDFS y ZooKeeper

Figura 6-43 muestra la relación entre ZooKeeper y HDFS.

Figura 6-43 Relación entre ZooKeeper y HDFS



Como cliente de un clúster de ZooKeeper, ZKFailoverController (ZKFC) supervisa el estado de NameNode. ZKFC solo se implementa en el nodo donde reside NameNode y en el NameNodes HDFS activo y en espera.

1. El ZKFC se conecta a ZooKeeper y guarda información como nombres de host en ZooKeeper bajo el directorio de znode **/hadoop-ha**. NameNode que crea el directorio primero se considera como el nodo activo, y el otro es el nodo en espera. NameNodes lee la información de NameNode periódicamente a través de ZooKeeper.
2. Cuando el proceso del nodo activo finaliza de forma anormal, NameNode en espera detecta cambios en el directorio **/hadoop-ha** a través de ZooKeeper y, a continuación, se hace cargo del servicio del NameNode activo.

6.9.4 Funciones mejoradas de código abierto de HDFS

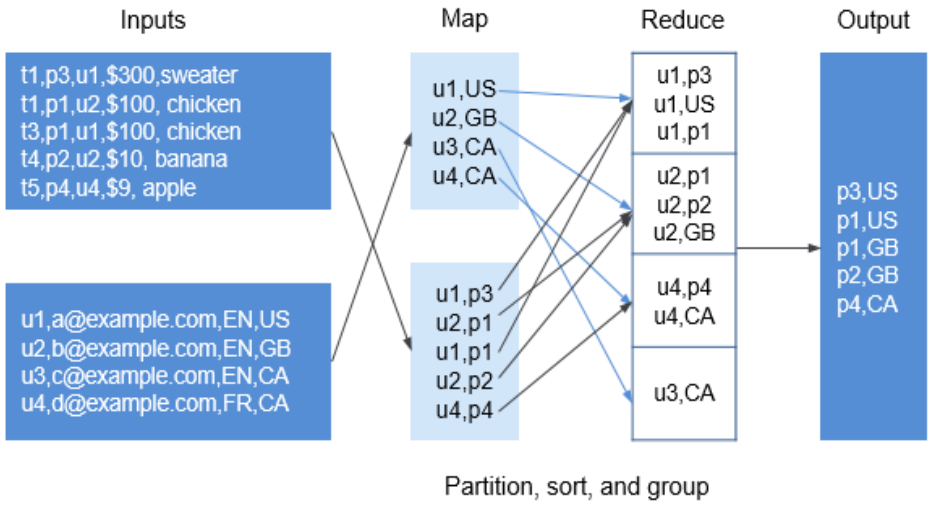
Función de código abierto mejorada: Colocación de bloques de archivos

En el escenario de resumen de datos y estadísticas fuera de línea, Join es una función informática de uso frecuente, y se implementa en el MapReduce de la siguiente manera:

1. La tarea de Map procesa los registros de los dos archivos de tabla en Join Key y Value, realiza la partición hash por Join Key y envía los datos a diferentes tareas de Reduce para su procesamiento.
2. Las tareas de Reduce que leen datos en la tabla izquierda recursivamente en el modo de nested loop y recorra cada línea de la tabla derecha. Si los valores de join key son idénticos, los resultados de join se generan.

   El método anterior reduce drásticamente el rendimiento del cálculo de join. Debido a que se requiere una gran cantidad de transferencia de datos de red cuando los datos almacenados en diferentes nodos se envían desde MAP a Reduce, como se muestra en **Figura 6-44**.

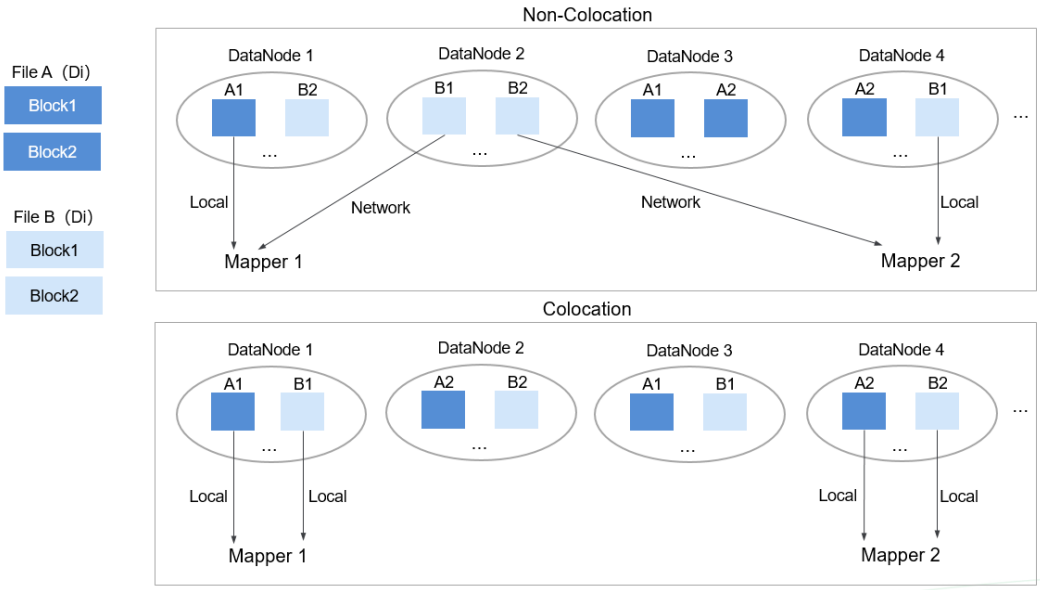
Figura 6-44 Transmisión de datos en el escenario de no colocación



Las tablas de datos se almacenan en el sistema de archivos físico por bloque de HDFS. Por lo tanto, si dos bloques a join se colocan en el mismo host en consecuencia después de que se particionen por join key, puede obtener los resultados directamente desde Map join en el nodo local sin ninguna transferencia de datos en el proceso Reduce del cálculo de join. Esto mejorará en gran medida el rendimiento.

Con la característica de distribución idéntica de los datos de HDFS, se asigna un mismo ID de distribución a los archivos, FileA y FileB, en los que se deben realizar cálculos de asociación y suma. De esta manera, todos los bloques se distribuyen juntos, y el cálculo se puede realizar sin recuperar datos entre nodos, lo que mejora enormemente el rendimiento de join de MapReduce.

Figura 6-45 Distribución de bloques de datos en escenarios de colocación y no colocación

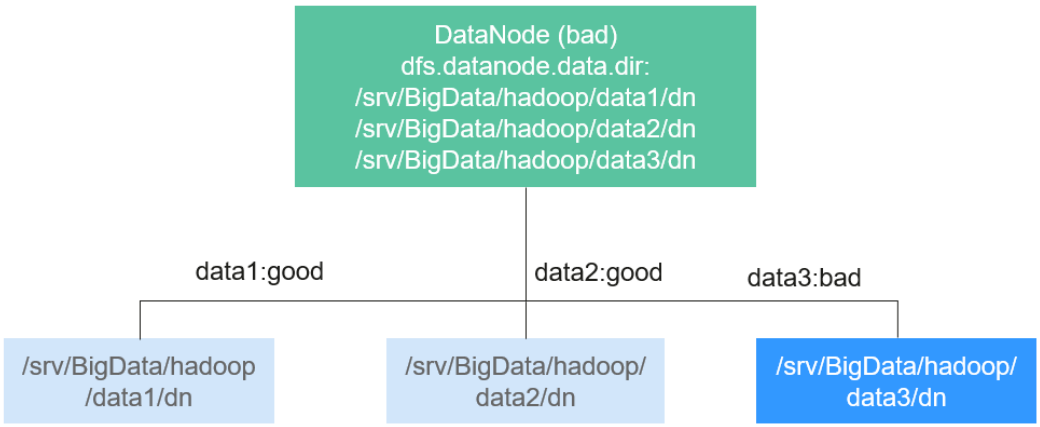


Función de código abierto mejorada: Configuración de volumen de disco duro dañado

En la versión de código abierto, si se configuran varios volúmenes de almacenamiento de datos para un DataNode, el DataNode deja de proporcionar servicios de forma predeterminada si uno de los volúmenes está dañado. Si el elemento de configuración **dfs.DataNode.failed.volumes.tolerated** está configurado para especificar el número de volúmenes dañados permitidos, el DataNode sigue proporcionando servicios cuando el número de volúmenes dañados no supera el umbral.

El valor de **dfs.DataNode.failed.volumes.tolerated** oscila entre -1 y el número de volúmenes de disco configurados en el DataNode. El valor predeterminado es **-1**, como se muestra en el documento **Figura 6-46**.

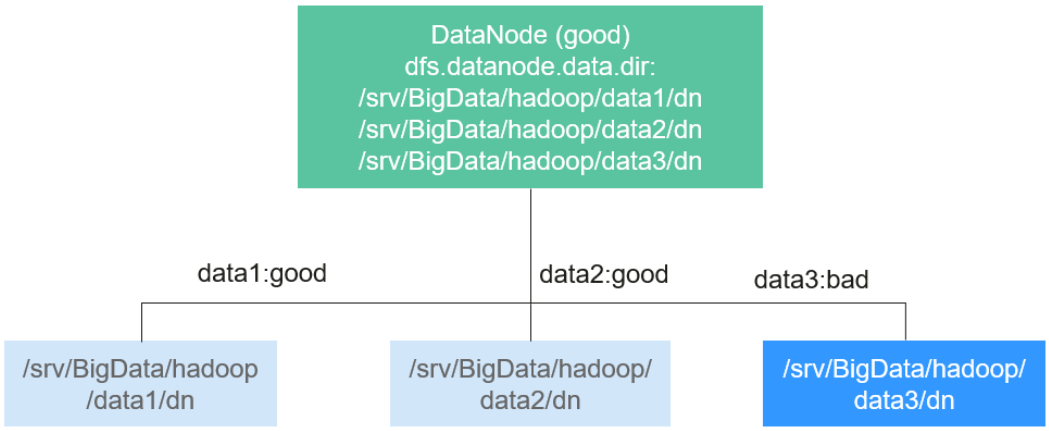
Figura 6-46 Elemento que se establece en 0



Por ejemplo, se montan tres volúmenes de almacenamiento de datos en un DataNode y el **dfs.DataNode.failed.volumes.tolerated** se establece en 1. En este caso, si un volumen de

almacenamiento de datos del DataNode no está disponible, este DataNode todavía puede proporcionar servicios, como se muestra en **Figura 6-47**.

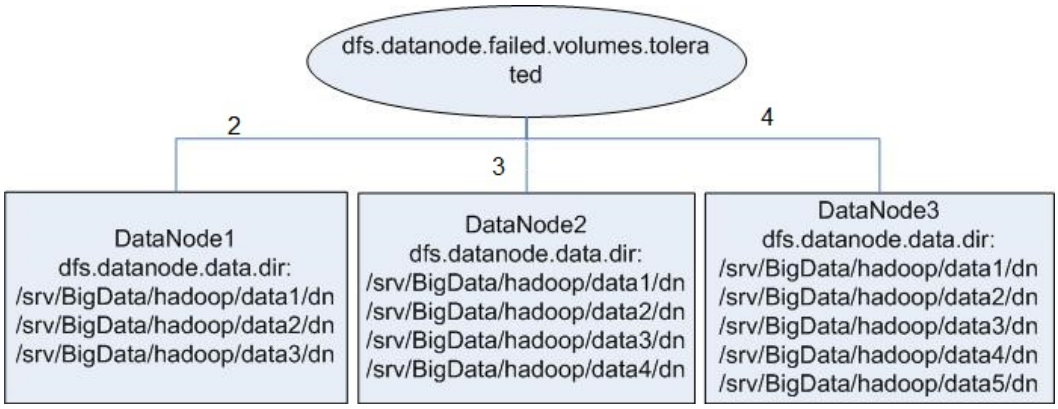
Figura 6-47 Elemento que se establece en 1



Este elemento de configuración nativo tiene algunos defectos. Cuando el número de volúmenes de almacenamiento de datos en cada DataNode es inconsistente, debe configurar cada DataNode de forma independiente en lugar de generar el archivo de configuración unificado para todos los nodos.

Supongamos que hay tres DataNodes en un clúster. El primer nodo tiene tres directorios de datos, el segundo nodo tiene cuatro y el tercer nodo tiene cinco. Si desea asegurarse de que los servicios de DataNode están disponibles cuando solo hay un directorio de datos disponible, debe realizar la configuración como se muestra en **Figura 6-48**.

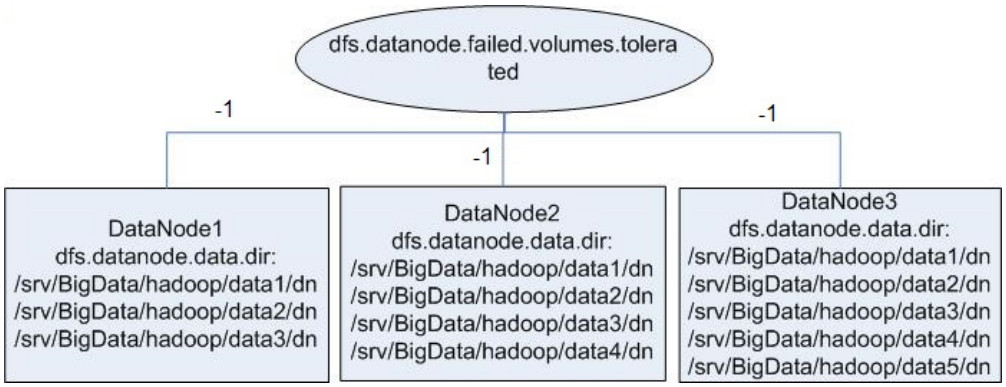
Figura 6-48 Configuración de atributos antes de ser mejorado



En HDFS mejorados de desarrollo propio, este elemento de configuración se mejora, con un valor agregado **-1**. Cuando este elemento de configuración se establece en **-1**, todos los DataNodes pueden proporcionar servicios siempre que esté disponible un volumen de almacenamiento de datos en todos los DataNodes.

Para resolver el problema en el ejemplo anterior, establezca esta configuración en **-1** como se muestra en **Figura 6-49**.

Figura 6-49 Configuración de atributos después de ser mejorado



Función de código abierto mejorada: aceleración de inicio de HDFS

En HDFS, cuando se inicia NameNodes, es necesario cargar el archivo de metadatos FsImage. A continuación, el DataNodes reportará la información del bloque de datos después del inicio del DataNodes. Cuando la información de bloque de datos reportada por DataNodes alcanza el porcentaje preestablecido, el NameNodes sale del modo seguro para completar el proceso de inicio. Si el número de archivos almacenados en el HDFS alcanza el nivel de millones o mil millones, los dos procesos consumen mucho tiempo y darán lugar a un largo tiempo de inicio del NameNode. Por lo tanto, esta versión optimiza el proceso de carga del archivo de metadatos FsImage.

En HDFS de código abierto, FsImage almacena todo tipo de información de metadatos. Cada tipo de información de metadatos (como la información de metadatos de archivos y la información de metadatos de carpetas) se almacena en un bloque de sección, respectivamente. Estos bloques de sección se cargan en modo serie durante el inicio. Si se almacena un gran número de archivos y carpetas en el HDFS, la carga de las dos secciones requiere mucho tiempo, lo que prolonga el tiempo de inicio del HDFS. El NameNode HDFS divide cada tipo de metadatos por segmentos y almacena los datos en varias secciones al generar los archivos FsImage. Cuando se inicia el NameNodes, las secciones se cargan en modo paralelo. Esto acelera el inicio de HDFS.

Función de código abierto mejorada: Políticas de colocación de bloques basadas en etiquetas (HDFS Nodelabel)

Debe configurar los nodos para almacenar bloques de datos de archivos HDFS en función de las características de datos. Puede configurar una expresión de etiqueta en un directorio o archivo HDFS y asignar una o más etiquetas a un DataNode para que los bloques de datos de archivo se puedan almacenar en DataNodes especificado. Si se utiliza la política de colocación de bloques de datos basada en etiquetas para seleccionar DataNodes para almacenar los archivos especificados, el intervalo de DataNode se especifica en función de la expresión de etiqueta. A continuación, se seleccionan los nodos adecuados del intervalo especificado.

- Puede almacenar las réplicas de bloques de datos en los nodos con diferentes etiquetas en consecuencia. Por ejemplo, almacenar dos réplicas del bloque de datos en el nodo etiquetado con L1, y almacenar otras réplicas del bloque de datos en los nodos etiquetados con L2.
- Puede establecer la política en caso de fallo de colocación de bloques, por ejemplo, seleccionar un nodo de todos los nodos al azar.

Figura 6-50 da un ejemplo:

- Los datos en **/HBase** se almacenan en A, B y D.
- Los datos en **/Spark** se almacenan en A, B, D, E y F.
- Los datos en **/user** se almacenan en C, D y F.
- Los datos en **/user/shl** se almacenan en A, E y F.

Figura 6-50 Ejemplo de política de colocación de bloques basada en etiquetas

Función de código abierto mejorada: balanceo de carga HDFS

Las políticas de lectura y escritura actuales de HDFS son principalmente para la optimización local sin considerar la carga real de nodos o discos. Basado en cargas de E/S de diferentes nodos, el balanceo de carga de HDFS garantiza que cuando se realizan operaciones de lectura y escritura en el cliente HDFS, el nodo con baja carga de E/S se selecciona para realizar tales operaciones para balancear la carga de E/S y utilizar completamente el rendimiento global del grupo.

Si balancel de carga de HDFS está habilitado durante la escritura de archivos, el NameNode selecciona un DataNode (en el orden de nodo local, rack local y rack remoto). Si la carga de E/S del nodo seleccionado es pesada, el NameNode elegirá otro DataNode con una carga más ligera.

Si HDFS Load Balance está habilitado durante la lectura de archivos, un cliente HDFS envía una solicitud al NameNode para proporcionar la lista de DataNodes que almacenan el bloque

que se va a leer. El NameNode devuelve una lista de DataNodes ordenados por distancia en la topología de red. Con la función de balanceo de carga HDFS, los DataNodes de la lista también se ordenan por su carga de E/S. Los DataNodes con carga pesada se encuentran en la parte inferior de la lista.

Función mejorada de código abierto: movimiento automático de datos HDFS

Hadoop se ha utilizado para el procesamiento por lotes de datos inmensos en mucho tiempo. El modelo HDFS existente se utiliza para satisfacer las necesidades de las aplicaciones de procesamiento por lotes muy bien porque tales aplicaciones se centran más en el rendimiento que en el retardo.

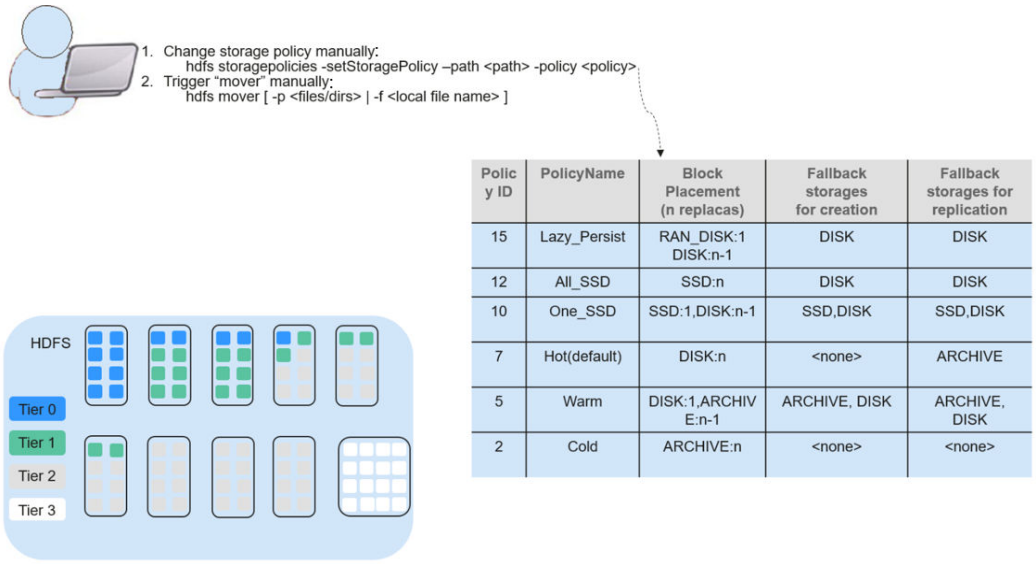
Sin embargo, como Hadoop se utiliza cada vez más para aplicaciones de capa superior que requieren acceso aleatorio frecuente de E/S, como Hive y HBase, los discos de baja latencia, como el disco de estado sólido (SSD), se ven favorecidos en escenarios sensibles al retardo. Para satisfacer la tendencia, HDFS admite una variedad de tipos de almacenamiento. Los usuarios pueden elegir un tipo de almacenamiento de acuerdo a sus necesidades.

Las políticas de almacenamiento varían en función de la frecuencia con la que se utilicen los datos. Por ejemplo, si los datos a los que se accede con frecuencia en HDFS se marcan como **ALL_SSD** o **HOT** los datos a los que se accede varias veces pueden marcarse como **WARM** y los datos a los que se accede raramente (solo una o dos veces) pueden marcarse como **COLD**. Puede seleccionar diferentes políticas de almacenamiento de datos según la frecuencia de acceso a los datos.

Sin embargo, los discos de baja latencia son mucho más caros que los discos giratorios. Los datos suelen ver un uso inicial intenso con una disminución en el uso durante un período de tiempo. Por lo tanto, puede ser útil si los datos que ya no se utilizan se mueven de discos costosos a los medios de almacenamiento más baratos.

Un ejemplo típico es el almacenamiento de registros de detalle. Los nuevos registros de detalle se importan a SSD porque las aplicaciones de capa superior consultan con frecuencia. A medida que disminuye la frecuencia de acceso a estos registros detallados, se trasladan a un almacenamiento más barato.

Antes de lograr el movimiento automático de datos, debe determinar manualmente por tipo de servicio si los datos se utilizan con frecuencia, establecer manualmente una política de almacenamiento de datos y activar manualmente la Herramienta de movimiento automático de datos HDFS, como se muestra en la siguiente figura.



Policy ID	PolicyName	Block Placement (n replicas)	Fallback storages for creation	Fallback storages for replication
15	Lazy_Persist	RAN_DISK:1 DISK:n-1	DISK	DISK
12	All_SSD	SSD:n	DISK	DISK
10	One_SSD	SSD:1,DISK:n-1	SSD,DISK	SSD,DISK
7	Hot(default)	DISK:n	<none>	ARCHIVE
5	Warm	DISK:1,ARCHIVE:n-1	ARCHIVE, DISK	ARCHIVE, DISK
2	Cold	ARCHIVE:n	<none>	<none>

Si los datos antiguos se pueden identificar automáticamente y transferir a un almacenamiento más barato (como el disco/archivo), verá reducciones significativas de costos y mejoras en la eficiencia de la gestión de datos.

La herramienta de movimiento automático de datos HDFS es el núcleo del movimiento automático de datos HDFS. Establece automáticamente una política de almacenamiento en función de la frecuencia con la que se utilicen los datos. Específicamente, las funciones de la herramienta de movimiento automático de datos HDFS pueden:

- Marcar una política de almacenamiento de datos como **All_SSD**, **One_SSD**, **Hot**, **Warm**, **Cold** o **FROZEN** según la edad, el tiempo de acceso y las reglas de movimiento manual de datos.
- Defina reglas para distinguir los datos fríos y calientes según age de los datos, access time y las reglas de migración manual.
- Definir las medidas que se deben tomar si se cumplen las normas basadas en la age.

  MARK: la acción para identificar si los datos se usan con frecuencia o rara vez en función de las reglas de age y establecer una política de almacenamiento de datos.
  MOVE: la acción para invocar la Herramienta de movimiento automático de datos HDFS y mover datos según las reglas de age para identificar si los datos se utilizan con frecuencia o rara vez después de haber determinado la política de almacenamiento correspondiente.
  - **MARK**: identifica si los datos se utilizan con frecuencia o con poca frecuencia y establece la política de almacenamiento de datos.
  - **MOVE**: la acción para invocar la Herramienta de movimiento automático de datos HDFS y mover datos entre niveles.
  - **SET_REPL**: la acción para establecer una nueva cantidad de réplica para un archivo.
  - **MOVE_TO_FOLDER**: la acción para mover archivos a una carpeta de destino.
  - **DELETE**: la acción para eliminar un archivo o directorio.
  - **SET_NODE_LABEL**: la acción para establecer etiquetas de nodo de un archivo.

Con la función de movimiento automático de datos HDFS, solo tiene que definir age en función de las reglas de access time. La herramienta de movimiento automático de datos HDFS hace coincidir los datos de acuerdo con las reglas basadas en age, establece políticas de

almacenamiento y mueve datos. De esta manera, se mejora la eficiencia de la gestión de datos y la eficiencia de los recursos del clúster.

6.10 HetuEngine

6.10.1 Descripción de producto de HetuEngine

Esta sección solo se aplica a MRS 3.1.2-LTS.3.

Descripción de HetuEngine

HetuEngine es un motor interno de virtualización de datos y análisis SQL interactivo de alto rendimiento. Se integra perfectamente con el ecosistema de big data para implementar consultas interactivas de cantidades masivas de datos en cuestión de segundos, y admite el acceso unificado a datos entre fuentes y dominios para permitir el análisis de convergencia SQL en un solo lugar en el lago de datos, entre lagos, y entre las casas del lago.

Arquitectura de HetuEngine

HetuEngine consta de diferentes módulos. **Figura 6-51** muestra la arquitectura.

Figura 6-51 Arquitectura de HetuEngine

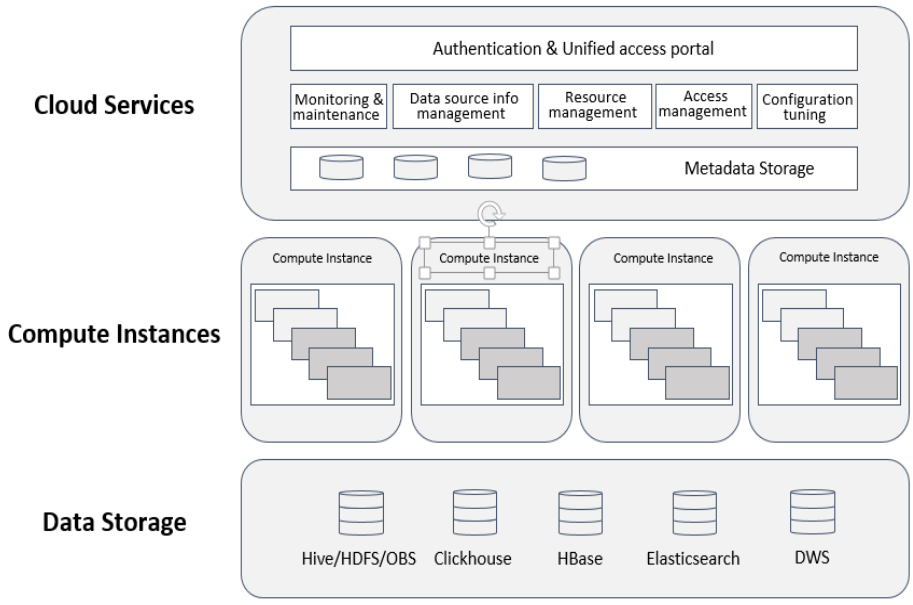


Tabla 6-10 Descripción del módulo

Módulo	Concepto	Descripción
Cloud serv	HetuEngine CLI/ JDBC	Cliente de HetuEngine, a través del cual se envía la solicitud de consulta y se devuelven y muestran los resultados.

Módulo	Concepto	Descripción
ice layer	HSBroker	Componente de gestión de servicios de HetuEngine. Gestiona y verifica las instancias de cómputo, supervisa el estado de salud y realiza el mantenimiento automático.
	HSConsole	Proporciona GUI de operación visualizada y RESTful APIs para la gestión de la información de origen de datos, la gestión de instancias de cómputo y la consulta automática de tareas.
	HSFabric	Proporciona transferencia de datos segura y de alto rendimiento entre dominios (centros de datos).
Engine layer	Coordinator	Nodo de gestión de instancias informáticas de HetuEngine. Recibe y analiza sentencias de SQL, genera y optimiza planes de ejecución, asigna tareas y programa recursos.
	Worker	Nodo de trabajo de las instancias informáticas de HetuEngine. Proporciona capacidades tales como la extracción de datos paralelos de fuentes de datos y la computación SQL distribuida.

Escenarios de aplicación de HetuEngine

HetuEngine admite consultas conjuntas rápidas entre orígenes (múltiples fuentes de datos, como Hive, HBase, GaussDB(DWS), Elasticsearch y ClickHouse) y entre dominios (múltiples regiones o centros de datos), especialmente para consultas rápidas interactivas de datos de Hive y Hudi en el clúster de Hadoop (MRS).

6.10.2 Relación entre HetuEngine y otros componentes

La instalación de HetuEngine depende del clúster MRS. **Tabla 6-11** enumera los componentes de los que depende la instalación de HetuServer.

Tabla 6-11 Componentes de los que depende HetuEngine

Nombre	Descripción
HDFS	Hadoop Distributed File System, compatible con el acceso a datos de alto rendimiento y adecuado para aplicaciones con conjuntos de datos a gran escala.
Hive	Almacén de datos de código abierto basado en Hadoop. Almacena datos estructurados e implementa análisis de datos básicos utilizando el lenguaje de consultas de Hive (HQL), un lenguaje similar a SQL.
ZooKeeper	Permite una coordinación distribuida altamente confiable. Ayuda a prevenir las fallas de punto único (SPOF) y proporciona servicios confiables para las aplicaciones.

Nombre	Descripción
KrbServer	Centro de gestión de claves que distribuye facturas.
Yarn	Sistema de gestión de recursos, que es un módulo general de recursos que gestiona y programa recursos para diversas aplicaciones.
DBService	DBService es un sistema de almacenamiento de base de datos relacional de alta disponibilidad que proporciona funciones de copia de respaldo y restauración de metadatos.

6.11 Hive

6.11.1 Principios básicos de Hive

Hive es una infraestructura de almacenamiento de datos basada en Hadoop. Proporciona una serie de herramientas para extraer, transformar y cargar (ETL) datos. Hive es un mecanismo que puede almacenar, consultar y analizar datos masivos almacenados en Hadoop. Hive define un lenguaje de consulta simple similar a SQL, que se conoce como HiveQL. Permite a los usuarios familiarizados con SQL consultar datos. El cálculo de datos de Hive depende de MapReduce, Spark, y Tez.

El nuevo motor de ejecución Tez se utiliza para reemplazar el MapReduce original, mejorando significativamente el rendimiento. Tez puede convertir varios trabajos dependientes en un trabajo, por lo que solo una vez que se requiere la escritura HDFS y se necesitan menos nodos de tránsito, lo que mejora enormemente el rendimiento de los trabajos DAG.

Hive proporciona las siguientes funciones:

- Analiza datos estructurados masivos y resume los resultados del análisis.
- Permite compilar trabajos complejos de MapReduce en lenguajes SQL.
- Soporta formatos flexibles de almacenamiento de datos, incluidos JavaScript object notation (JSON), comma separated values (CSV), TextFile, RCFile, SequenceFile, y ORC (Optimized Row Columnar).

Estructura del sistema de Hive:

- Interfaz de usuario: Hay tres interfaces de usuario disponibles, es decir, CLI, Client y WUI. CLI es la interfaz de usuario más utilizada. Se inicia una transcripción de Hive cuando se inicia CLI. Client hace referencia a un cliente de Hive y un usuario de cliente se conecta a Hive Server. Al entrar en el modo cliente, debe especificar el nodo donde reside el servidor Hive e iniciar el servidor Hive en este nodo. La web UI se utiliza para acceder a Hive a través de un navegador. MRS solo puede acceder a Hive en modo cliente. Para obtener más información, consulte **Uso de Hive desde Scratch**Para obtener más información sobre cómo desarrollar aplicaciones Hive, consulte **Desarrollo de aplicaciones de Hive**.
- Almacenamiento de metadatos: Hive almacena metadatos en bases de datos, por ejemplo, MySQL y Derby. Los metadatos de Hive incluyen un nombre de tabla, columnas y particiones de tabla y sus propiedades, propiedades de tabla (que indican si una tabla es una tabla externa) y el directorio donde se almacenan los datos de la tabla.

Marco de Hive

Hive es un proceso de servicio de instancia única que proporciona servicios al traducir HQL en trabajos de MapReduce u operaciones HDFS relacionadas. **Figura 6-52** muestra cómo Hive está conectado a otros componentes.

Figura 6-52 Marco de Hive

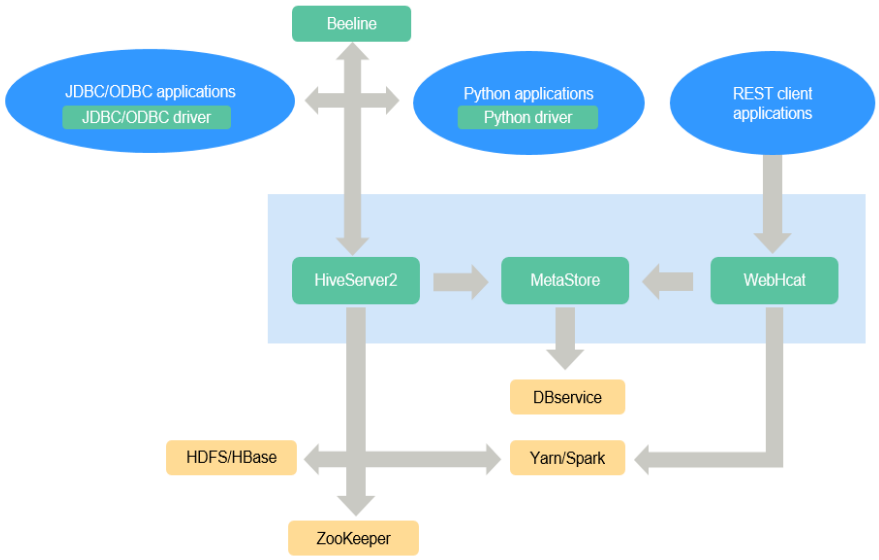


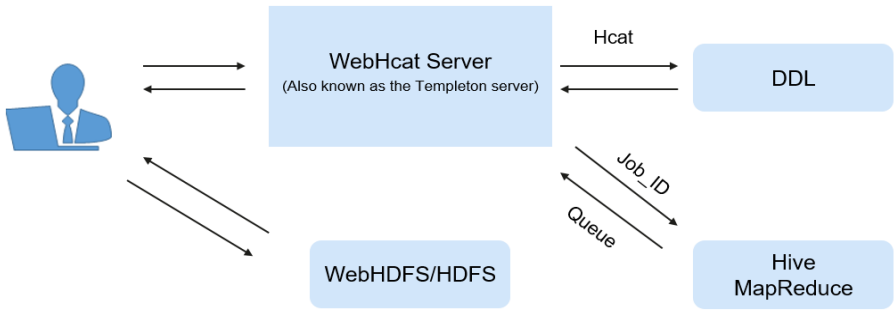
Tabla 6-12 Descripción del módulo

Módulo	Descripción
HiveServer	Se pueden implementar varias HiveServers en un clúster para compartir cargas. HiveServer proporciona servicios de base de datos Hive externamente, traduce sentencias HQL en tareas YARN u operaciones HDFS relacionadas para completar la extracción, conversión y análisis de datos.
MetaStore	● Se pueden desplegar varias MetaStores en un clúster para compartir cargas. MetaStore proporciona servicios de metadatos de Hive, así como lee, escribe, mantiene y modifica la estructura y las propiedades de las tablas de Hive. ● MetaStore proporciona API de Thrift para HiveServer, Spark, WebHCat y otros clientes de MetaStore para acceder y operar metadatos.
WebHCat	Se pueden desplegar varios WebHCats en un clúster para compartir cargas. WebHCat proporciona las API de REST y ejecuta los comandos Hive a través de las API de REST para enviar trabajos de MapReduce.
Hive client	El cliente de Hive incluye la interfaz de línea de comandos de hombre-máquina (CLI) Beeline, controlador JDBC para aplicaciones de JDBC, controlador Python para aplicaciones de Python y archivos de HCatalog JAR para MapReduce.

Módulo	Descripción
ZooKeeper cluster	Como nodo temporal, ZooKeeper registra la lista de direcciones IP de cada instancia de HiveServer. El controlador de cliente se conecta a ZooKeeper para obtener la lista y selecciona las instancias de HiveServer correspondientes en función del mecanismo de enrutamiento.
HDFS/HBase cluster	El clúster de HDFS almacena los datos de la tabla Hive.
MapReduce/ YARN cluster	Proporciona servicios informáticos distribuidos. La mayoría de las operaciones de datos de Hive dependen de MapReduce. La función principal de HiveServer es traducir declaraciones HQL en trabajos MapReduce para procesar datos masivos.

HCatalog se basa en Hive Metastore e incorpora la capacidad de DDL de Hive. HCatalog también es una capa de gestión de almacenamiento y tablas basada en Hadoop que permite la lectura y escritura de datos convenientes en tablas de HDFS mediante el uso de diferentes herramientas de procesamiento de datos como Pig y MapReduce. Además, HCatalog también proporciona API de lectura/escritura para estas herramientas y utiliza una CLI de Hive para publicar comandos para definir datos y consultar metadatos. Después de encapsular estos comandos, WebHCat Server puede proporcionar las API de RESTful, como se muestra en **Figura 6-53**.

Figura 6-53 Arquitectura lógica de WebHCat



Principios

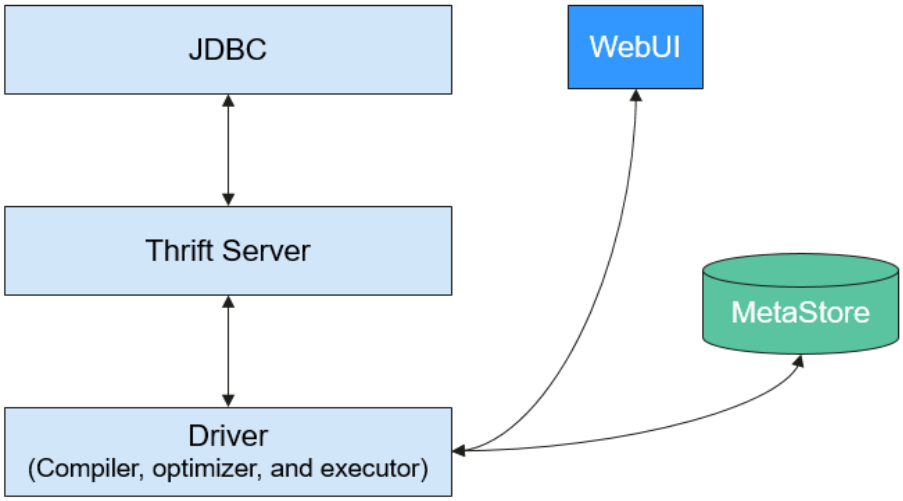
Hive funciona como un almacén de datos basado en la arquitectura de HDFS y MapReduce y traduce las sentencias HQL en trabajos de MapReduce u operaciones de HDFS. Para obtener más información sobre Hive y HQL, consulte **Manual de lenguaje de HiveQL**.

Figura 6-54 muestra la estructura de Hive.

- **Metastore**: lee, escribe y actualiza metadatos como tablas, columnas y particiones. Su capa inferior son las bases de datos relacionales.
- **Driver**: gestiona el ciclo de vida de la ejecución de HiveQL y participa en toda la ejecución del trabajo de Hive.
- **Compiler**: traduce sentencias HQL en una serie de trabajos de Map o Reduce interdependientes.

- **Optimizer**: se clasifica en optimizador lógico y optimizador físico para optimizar los planes de ejecución de HQL y los trabajos de MapReduce respectivamente.
- **Executor**: ejecuta trabajajos de Map o Reduce en función de las dependencias del trabajo.
- **ThriftServer**: funciona como los servidores de JDBC, proporciona API de Thrift y se integra con Hive y otras aplicaciones.
- **Clients**: incluyen las API de WebUI y JDBC y proporciona API para el acceso de los usuarios.

Figura 6-54 Marco de Hive



6.11.2 Principios de Hive CBO

Principios de Hive CBO

CBO es la abreviatura de Optimización Basada en Costos.

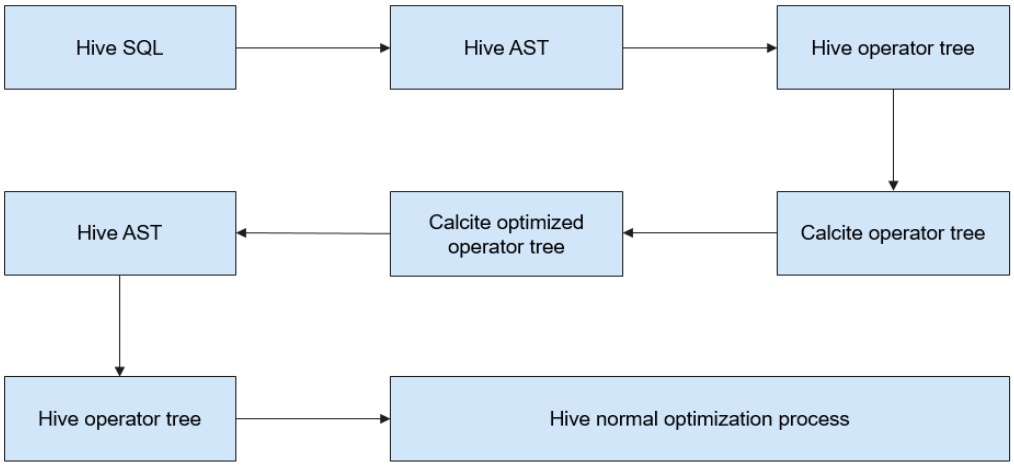
Optimizará lo siguiente:

Durante la compilación, el CBO calcula la secuencia de unión más eficiente basándose en tablas y condiciones de consulta implicadas en las sentencias de consulta para reducir el tiempo y los recursos necesarios para la consulta.

En Hive, la CBO se implementa de la siguiente manera:

Hive utiliza el componente de código abierto Apache Calcite para implementar la CBO. Las sentencias SQL se convierten primero en Hive Abstract Syntax Trees (ASTs) y luego en RelNodes que pueden ser identificados por Calcita. Después de que Calcita ajusta la secuencia de unión de RelNodes, RelNodes se convierten en ASTs por Hive para continuar la optimización lógica y física. **Figura 6-55** muestra el flujo de trabajo.

Figura 6-55 Proceso de implementación de CBO



Calcita ajusta la secuencia de unión de la siguiente manera:

1. Una tabla se selecciona como la primera tabla de las tablas que se van a unir.
2. Las tablas segunda y tercera se seleccionan en función del costo. De esta manera, se obtienen múltiples planes de ejecución diferentes.
3. Se calcula un plan con los costes mínimos y sirve como la secuencia final.

El método de cálculo de costes es el siguiente:

En la versión actual, los costes se miden en función del número de entradas de datos tras la unión. Menos entradas de datos significan menos costo. El número de entradas de datos unidas depende de la tasa de selección de las tablas unidas. El número de entradas de datos en una tabla se obtiene basándose en las estadísticas a nivel de tabla.

El número de entradas de datos en una tabla después del filtrado se estima en función de las estadísticas a nivel de columna, incluidos los valores máximos (max), valores mínimos (min) y Número de valores distintos (NDV).

Por ejemplo, hay una tabla **table_a** cuyo número total de registros de datos es de 1,000,000 y NDV es de 50. Las condiciones de consulta son las siguientes:

```
Select * from table_a where colum_a='value1';
```

El número estimado de entradas de datos consultadas es: 1,000,000 x 1/50 = 20,000. La tasa de selección es del 2%.

A continuación se toma el TPC-DS Q3 como ejemplo para describir cómo el CBO ajusta la secuencia de join:

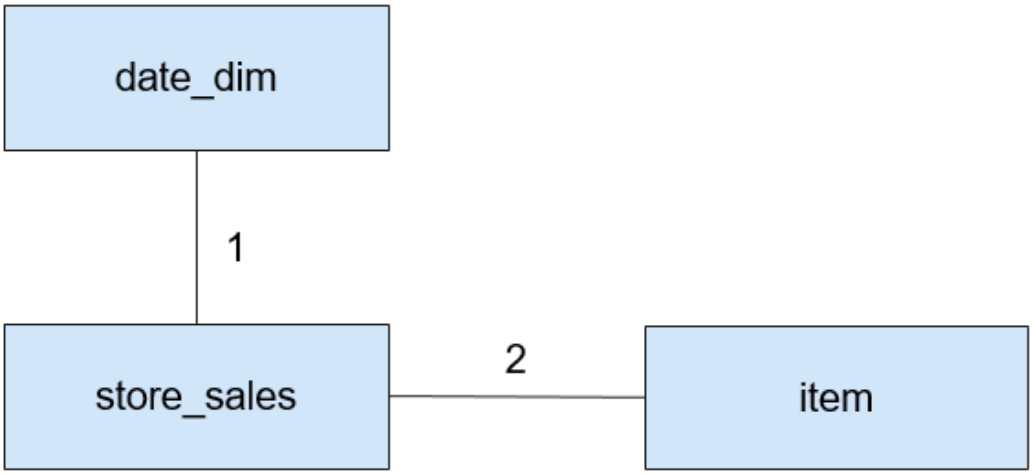
```
select
    dt.d_year,
    item.i_brand_id brand_id,
    item.i_brand brand,
    sum(ss_ext_sales_price) sum_agg
from
    date_dim dt,
    store_sales,
    item
where
    dt.d_date_sk = store_sales.ss_sold_date_sk
    and store_sales.ss_item_sk = item.i_item_sk
```



```
    and item.i_manufact_id = 436
    and dt.d_moy = 12
group by dt.d_year , item.i_brand , item.i_brand_id
order by dt.d_year , sum_agg desc , brand_id
limit 10;
```

Explicación de la sentencia: Esta sentencia indica que la unión interna se realiza para tres tablas: la tabla **store_sales** es una tabla de hechos con alrededor de 2,900,000,000 entradas de datos, la tabla **date_dim** es una tabla de dimensiones con alrededor de 73,000 entradas de datos y la tabla **item** es una tabla de dimensiones con alrededor de 18,000 entradas de datos. Cada tabla tiene condiciones de filtrado. **Figura 6-56** muestra la relación de join.

Figura 6-56 Relación de join



El CBO debe seleccionar primero las tablas que ofrecen el mejor efecto de filtrado para la unión.

Analizando min, max, NDV y el número de entradas de datos, el CBO estima las tasas de selección de diferentes tablas de dimensiones, como se muestra en **Tabla 6-13**.

Tabla 6-13 Filtrado de datos

Tabla	Número de entradas de datos originales	Número de entradas de datos después de filtrar	Tasa de selección
date_dim	73,000	6,200	8.5%
item	18,000	19	0.1%

La tasa de selección se puede estimar de la siguiente manera: Tasa de selección = Número de entradas de datos después del filtrado/Número de entradas de datos originales

Como se muestra en la tabla anterior, la tabla **item** tiene un mejor efecto de filtrado. Por lo tanto, el CBO se une a la tabla **item** primero antes de unirse a la tabla **date_dim**.

Figura 6-57 muestra el proceso de join cuando el CBO está deshabilitado.

Figura 6-57 Proceso de join cuando la CBO está deshabilitada

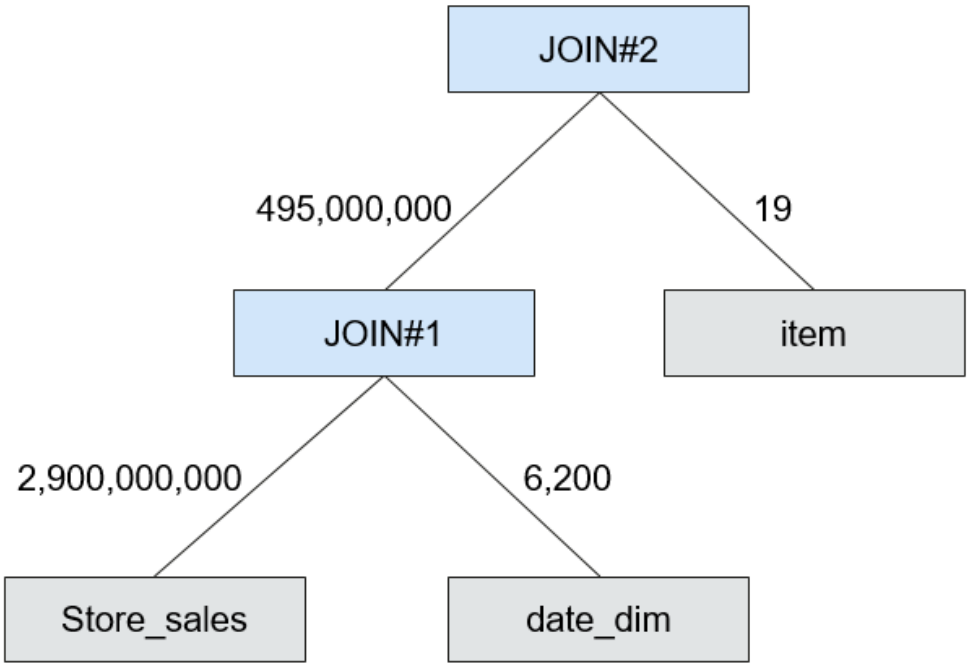
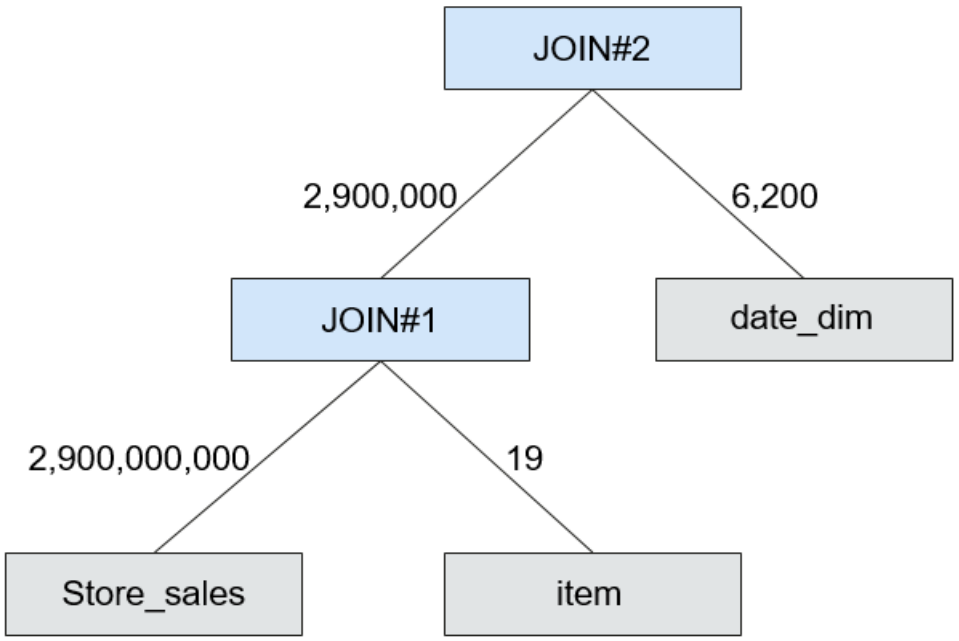


Figura 6-58 muestra el proceso de join cuando el CBO está habilitado.

Figura 6-58 Proceso de join cuando el CBO está habilitado



Después de habilitar el CBO, el número de entradas de datos intermedias se reduce de 495,000,000 a 2,900,000 y, por lo tanto, el tiempo de ejecución se puede reducir notablemente.

6.11.3 Relación entre Hive y otros componentes

Relación entre Hive y HDFS

Hive es un subproyecto de Apache Hadoop, que utiliza HDFS como sistema de almacenamiento de archivos. Analiza y procesa datos estructurados con almacenamiento subyacente altamente confiable soportado por HDFS. Todos los archivos de datos de la base de datos Hive se almacenan en HDFS, y todas las operaciones de datos en Hive también se realizan mediante API de HDFS.

Relación entre Hive y MapReduce

El cálculo de datos de Hive depende de MapReduce. MapReduce es también un subproyecto de Apache Hadoop y es un marco de computación paralelo basado en HDFS. Durante el análisis de datos, Hive analiza las sentencias de HQL enviadas por los usuarios en tareas de MapReduce y envía las tareas para que MapReduce las ejecute.

Relación entre Hive y Tez

Tez, un proyecto de código abierto de Apache, es un marco de computación distribuida que soporta gráficos acíclicos dirigidos (DAG). Cuando Hive utiliza el motor de Tez para analizar datos, analiza las sentencias de HQL enviadas por los usuarios en tareas de Tez y envía las tareas a Tez para su ejecución.

Relación entre Hive y DBService

MetaStore (servicio de metadatos) de Hive procesa la estructura y la información de atributos de los metadatos de Hive, como bases de datos, tablas y particiones de Hive. La información debe almacenarse en una base de datos relacional y es gestionada y procesada por MetaStore. En el producto, los metadatos de Hive son almacenados y mantenidos por el componente de DBService, y el servicio de metadatos es proporcionado por el componente de Metadata.

6.11.4 Función de código abierto mejorada

Función mejorada de código abierto: HDFS Colocation

HDFS Colocation es la función de control de ubicación de datos proporcionada por HDFS. La API de HDFS Colocation almacena datos asociados o datos en los que se realizan operaciones asociadas en el mismo nodo de almacenamiento.

Hive es compatible con HDFS Colocation. Cuando se crean tablas Hive, después de establecer la información del localizador para los archivos de tabla, los archivos de datos de las tablas relacionadas se almacenan en el mismo nodo de almacenamiento. Esto asegura una computación de datos conveniente y eficiente entre las tablas asociadas.

Función de código abierto mejorada: Cifrado de columnas

Hive admite encriptación de una o más columnas. Las columnas que se van a cifrar y el algoritmo de encriptación se pueden especificar cuando se crea una tabla Hive. Cuando se

insertan datos en la tabla utilizando la sentencia INSERT, las columnas relacionadas se cifran. El cifrado de columna Hive no admite vistas y el escenario Hive sobre HBase.

El mecanismo de cifrado de columnas de Hive admite dos algoritmos de cifrado que se pueden seleccionar para cumplir con los requisitos del sitio durante la creación de tablas:

- AES (la clase de encriptación es de **org.apache.hadoop.hive.serde2.AESRewriter**)
- SMS4 (la clase de encriptación es de **org.apache.hadoop.hive.serde2.SMS4Rewriter**)

Función de código abierto mejorada: eliminación de HBase

Debido a las limitaciones de los sistemas de almacenamiento subyacentes, Hive no admite la capacidad de eliminar una sola pieza de datos de tabla. En Hive on HBase, Hive en la solución de MRS admite la capacidad de eliminar una sola pieza de datos de tabla HBase. Utilizando una sintaxis específica, Hive puede eliminar uno o más fragmentos de datos de una tabla HBase.

Función de código abierto mejorada: delimitador de filas

En la mayoría de los casos, un carácter de retorno de carro se utiliza como delimitador de fila en las tablas de Hive almacenadas en archivos de texto, es decir, el carácter de retorno de carro se utiliza como el terminador de una fila durante las consultas.

Sin embargo, algunos archivos de datos están delimitados por caracteres especiales y no por un carácter de retorno de carro.

MRS Hive permite especificar diferentes caracteres o combinaciones de caracteres como delimitadores de filas para los datos de Hive en archivos de texto.

Función de código abierto mejorada: conmutación de REST API basada en HTTPS/HTTP

WebHCat proporciona REST APIs externas para Hive. De forma predeterminada, la versión de comunidad de código abierto utiliza el protocolo HTTP.

MRS Hive admite el protocolo HTTPS que es más seguro y permite la conmutación entre el protocolo HTTP y el protocolo HTTPS.

Función de código abierto mejorada: función de Transform

La función Transform no está permitida por Hive de la versión de código abierto. MRS Hive admite la configuración de la función Transform. La función está deshabilitada por defecto, que es la misma que la versión de la comunidad de código abierto.

Los usuarios pueden modificar las configuraciones de la función de Transform para habilitar la función. Sin embargo, existen riesgos de seguridad cuando se habilita la función Transform.

Función de código abierto mejorada: creación de funciones temporales sin permiso ADMIN

Debe tener permiso **ADMIN** al crear funciones temporales en Hive de la versión de comunidad de código abierto. MRS Hive admite la configuración de la función para crear funciones temporales con permiso **ADMIN**. La función está deshabilitada por defecto, que es la misma que la versión de la comunidad de código abierto.

Puede modificar las configuraciones de esta función. Una vez habilitada la función, puede crear funciones temporales sin permiso **ADMIN**.

Función de código abierto mejorada: Autorización de base de datos

En la versión de la comunidad de código abierto de Hive, solo el propietario de la base de datos puede crear tablas en la base de datos. MRS Hive puede otorgarle los permisos **CREATE** y **SELECT** en tablas en una base de datos. Después de que se le conceda el permiso para consultar datos en la base de datos, el sistema asocia automáticamente el permiso de consulta en todas las tablas de la base de datos.

Función de código abierto mejorada: Autorización de columna

La versión de la comunidad de código abierto de Hive solo admite el control de permisos a nivel de tabla. MRS Hive admite el control de permisos a nivel de columna. Se le pueden conceder permisos de columna, como **SELECT**, **INSERT** y **UPDATE**.

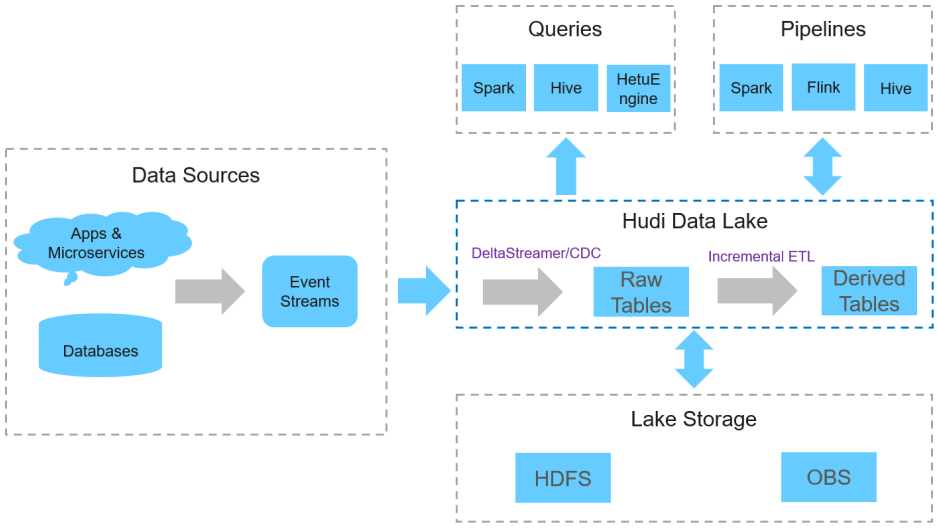
6.12 Hudi

Hudi es un formato de tabla de lago de datos que proporciona la capacidad de actualizar y eliminar datos, así como consumir nuevos datos en HDFS. Es compatible con múltiples motores de cómputo y proporciona interfaces de inserción, actualización y eliminación (IUD) y primitivas de transmisión, que incluyen upsert y extracción incremental, sobre conjuntos de datos en HDFS.

NOTA

Para utilizar Hudi, asegúrese de que el servicio Spark2x se ha instalado en el clúster de MRS.

Figura 6-59 Arquitectura básica de Hudi



Características

- La capacidad de transacción ACID admite la importación de datos en tiempo real al lago y la importación de datos por lotes al lago de datos.
- Las múltiples capacidades de visualización (vista de lectura optimizada/vista incremental/vista en tiempo real) permiten un análisis de datos rápido.

- El diseño de control de simultaneidad de múltiples versiones (MVCC) admite el retroceso de versiones de datos.
- La gestión automática de tamaños y diseños de archivos optimiza el rendimiento de las consultas y proporciona datos en tiempo casi real para las consultas.
- Se admiten lecturas y escrituras simultáneas. Los datos se pueden leer cuando se escriben basándose en el aislamiento de instantáneas.
- Bootstrapping es compatible para convertir tablas existentes en conjuntos de datos de Hudi.

Tecnologías clave y ventajas

- Mecanismo de índice conectable: Hudi proporciona múltiples mecanismos de índice para actualizar y eliminar rápidamente datos masivos.
- Compatibilidad con el ecosistema: Hudi admite varios motores de datos, incluidos Hive, Spark, HetuEngine, y Flink.

Dos tipos de tablas compatibles con Hudi

- Copy On Write

  Las tablas de Copy-on-write también se denominan tablas COW. Los archivos de parquet se utilizan para almacenar datos, y las operaciones de actualización interna deben realizarse reescribiendo los archivos de parquet originales.

  - Ventaja: Es eficiente porque solo se necesita leer un archivo de datos en la partición correspondiente.
  - Desventaja: Durante la escritura de datos, se necesita copiar una copia anterior y luego se genera un nuevo archivo de datos basado en la copia anterior. Este proceso requiere mucho tiempo. Por lo tanto, los datos leídos por la solicitud de lectura se retrasan.

- Merge On Read

  Las tablas de Merge-on-read también se denominan tablas MOR. La combinación de parquet basado en columnas y formato Avro basado en filas se utiliza para almacenar datos. Los archivos Parquet se utilizan para almacenar datos base, y los archivos Avro (también llamados archivos de registro) se utilizan para almacenar datos incrementales.

  - Ventaja: Los datos se escriben en delta log primero, y el tamaño de delta log es pequeño. Por lo tanto, el coste de escritura es bajo.
  - Desventaja: Los archivos deben compactarse periódicamente. De lo contrario, hay un gran número de archivos de fragmento. El rendimiento de lectura es pobre porque delta logs y los archivos de datos antiguos necesitan combinarse.

Hudi apoya tres tipos de vistas para las capacidades de lectura en diferentes escenarios

- Vista instantánea

  Proporciona los datos de instantánea más recientes de la tabla Hudi actual. Es decir, una vez que los últimos datos se escriben en la tabla Hudi, los datos recién escritos se pueden consultar a través de esta vista.

  Tanto las tablas COW como MOR soportan esta capacidad de vista.

- Vista incremental

Proporciona la capacidad de consulta incremental. Los datos incrementales después de una confirmación especificada pueden ser consultados. Esta vista se puede utilizar para extraer rápidamente datos incrementales.

Las tablas COW soportan esta capacidad de vista. Las tablas MOR también admiten esta capacidad de vista, pero la capacidad de vista incremental desaparece una vez que se realiza la operación compacta.

- Leer vista optimizada

  Proporciona únicamente los datos almacenados en el último archivo de Parquet.

  Esta vista es diferente para las tablas COW y MOR.

  Para las tablas COW, la capacidad de vista es la misma que la capacidad de vista en tiempo real. (Las tablas COW utilizan solo archivos de Parquet para almacenar datos.)

  Para las tablas MOR, sólo se accede a los archivos base, y se proporcionan los datos en los segmentos de archivo dados desde la última operación compacta. Se puede entender simplemente que esta vista proporciona solo los datos almacenados en los archivos de parquet de las tablas MOR, y los datos en los archivos de log se ignoran. Los datos proporcionados por esta vista pueden no ser los últimos. Sin embargo, una vez que se realiza la operación compacta en tablas MOR, los datos de log incrementales se fusionan en los datos base. En este caso, esta vista tiene la misma capacidad que la vista en tiempo real.

6.13 Hue

6.13.1 Principios básicos de Hue

Hue es un grupo de aplicaciones web que interactúan con componentes de big data de MRS. Le ayuda a explorar HDFS, realizar consultas de Hive e iniciar trabajos de MapReduce. Hue lleva aplicaciones que interactúan con todos los componentes de big data de MRS.

Hue proporciona las funciones del navegador de archivos y del editor de consultas:

- El navegador de archivos le permite navegar y operar directamente diferentes directorios de HDFS en la GUI.
- El editor de consultas puede escribir sentencias SQL simples para consultar datos almacenados en Hadoop, por ejemplo, HDFS, HBase y Hive. Con el editor de consultas, puede crear, gestionar y ejecutar sentencias de SQL fácilmente y descargar los resultados de la ejecución como un archivo de Excel.

En el WebUI proporcionado por Hue, puede realizar las siguientes operaciones en los componentes:

- HDFS:
  - Ver, crear, gestionar, renombrar, mover y eliminar archivos o directorios.
  - Cargar y descargar de archivos
  - Buscar archivos, directorios, propietarios de archivos y grupos de usuarios; cambiar los propietarios y permisos de los archivos y directorios.
  - Configurar manualmente las políticas de almacenamiento de directorios HDFS y las políticas de almacenamiento dinámico.
- Hive:

  - Editar y ejecutar sentencias de SQL/HQL. Guardar, copiar y editar la plantilla de SQL/HQL. Explicar sentencias de SQL/HQL. Guardar la sentencia de SQL/HQL y consultarla.
  - Presentación de la base de datos y presentación de la tabla de datos
  - Soportar diferentes tipos de almacenamiento de Hadoop
  - Utilizar MetaStore para agregar, eliminar, modificar y consultar bases de datos, tablas y vistas.

    NOTA

    Si se utiliza Internet Explorer para acceder a la página Hue para ejecutar sentencias de HiveSQL, la ejecución falla porque el navegador tiene problemas funcionales. Se recomienda utilizar un navegador compatible, por ejemplo, Google Chrome.

- Impala:
  - Editar y ejecutar sentencias de SQL/HQL. Guardar, copiar y editar la plantilla de SQL/HQL. Explicar sentencias de SQL/HQL. Guardar la sentencia de SQL/HQL y consultarla.
  - Presentación de la base de datos y presentación de la tabla de datos
  - Soportar diferentes tipos de almacenamiento de Hadoop
  - Utilizar MetaStore para agregar, eliminar, modificar y consultar bases de datos, tablas y vistas.

    NOTA

    Si se utiliza Internet Explorer para acceder a la página Hue para ejecutar sentencias de HiveSQL, la ejecución falla porque el navegador tiene problemas funcionales. Se recomienda utilizar un navegador compatible, por ejemplo, Google Chrome.
- MapReduce: compruebar las tareas de MapReduce que se están ejecutando o que se han finalizado en los clústeres, incluido su estado, la hora de inicio y finalización y logs de ejecución.
- Oozie: Hue proporciona la función gestor de trabajos Oozie, en este caso, puede usar Oozie en modo GUI.
- ZooKeeper: Hue proporciona la función del navegador de ZooKeeper para que utilice ZooKeeper en modo GUI.

Para obtener más información sobre Hue, consulte **https://gethue.com/**.

Arquitectura

Hue, que adopta el diseño MTV (Model-Template-View), es un programa de aplicación web que se ejecuta en Django Python. (Django Python es un marco de aplicaciones web que utiliza códigos de código abierto.)

Hue se compone de Proceso Supervisor y WebServer. Supervisor Process es el proceso principal de Hue que gestiona los procesos de aplicación. Supervisor Process y WebServer interactúan con aplicaciones en WebServer a través de las API de Thrift/REST, como se muestra en **Figura 6-60**.

Figura 6-60 Arquitectura de Hue

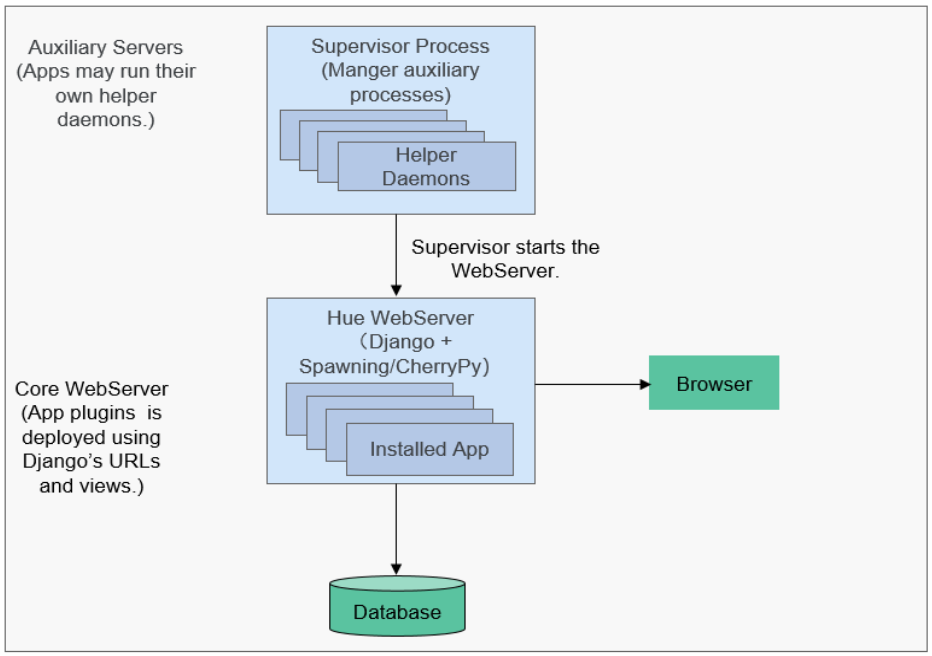


Tabla 6-14 describe los componentes mostrados en **Figura 6-60**.

Tabla 6-14 Descripción de la arquitectura

Nombre de la conexión	Descripción
Supervisor Process	Gestiona los procesos de las aplicaciones de WebServer, como iniciar, detener y supervisar los procesos.
Hue WebServer	Proporciona las siguientes funciones a través del marco de web de Django Python: ● Despliega aplicaciones. ● Proporciona la interfaz gráfica de usuario. ● Se conecta a bases de datos para almacenar datos persistentes de aplicaciones.

6.13.2 Relación entre Hue y otros componentes

Relación entre los clústeres de Hue y Hadoop

Figura 6-61 muestra cómo Hue interactúa con los clústeres de Hadoop.

Figura 6-61 Clústeres de Hue y Hadoop

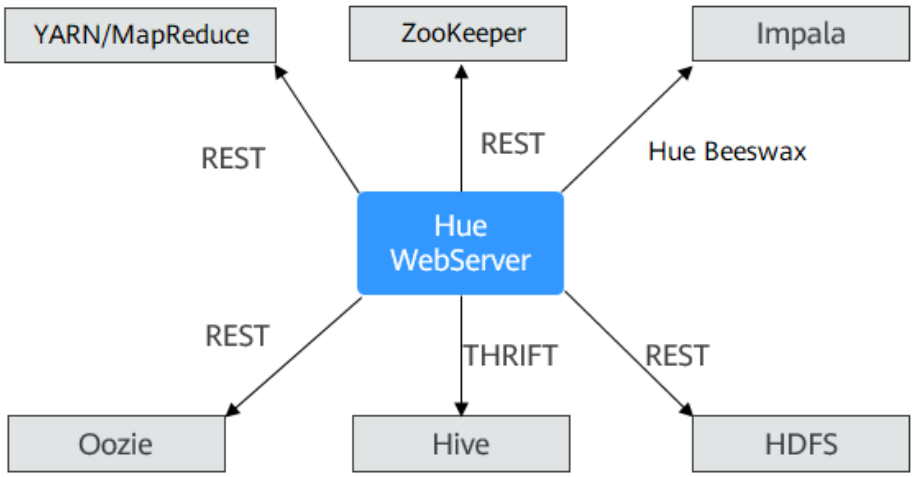


Tabla 6-15 Relación entre Hue y otros componentes

Nombre de la conexión	Descripción
HDFS	HDFS proporciona las API de REST para interactuar con Hue para consultar y operar archivos HDFS. Hue empaqueta una solicitud de usuario en los datos de la interfaz, envía la solicitud a HDFS a través de las API de REST y muestra los resultados de la ejecución en la interfaz de usuario web.
Hive	Hive proporciona interfaces de Thrift para interactuar con Hue, ejecutar sentencias de SQL de Hive y consultar metadatos de tablas. Si edita sentencias de HQL en la interfaz de usuario web de Hue, Hue envía las sentencias de HQL al servidor de Hive a través de las API de Thrift y muestra los resultados de ejecución en la interfaz de usuario web.
YARN/MapReduce	MapReduce proporciona las API de REST para interactuar con Hue y consultar información de trabajo YARN. Si va a la interfaz de usuario web de Hue, introduzca los parámetros de filtro, la interfaz de usuario envía los parámetros al fondo, y Hue invoca las API de REST proporcionadas por MapReduce (MR1/MR2-YARN) para obtener información como el estado de la tarea en ejecución, la hora de inicio/final, el registro de ejecución y más.
Oozie	Oozie proporciona las API de REST para interactuar con Hue, crear flujos de trabajo, coordinators y bundles, y gestionar y supervisar tareas. En la interfaz de usuario web de Hue se proporciona un flujo de trabajo gráfico, coordinator y un editor de bundle. Hue invoca las API de REST de Oozie para crear, modificar, eliminar, enviar y supervisar flujos de trabajo, coordinators y bundles.

Nombre de la conexión	Descripción
ZooKeeper	ZooKeeper proporciona las API de REST para interactuar con Hue y consultar información de nodo ZooKeeper. La información del nodo de ZooKeeper se muestra en la interfaz de usuario web de Hue. Hue invoca las API de REST de ZooKeeper para obtener la información del nodo.
Impala	Impala proporciona las API de Hue Beeswax para interactuar con Hue, ejecutar sentencias de Hive SQL y consultar metadatos de tablas. Si edita sentencias de HQL en la interfaz de usuario web de Hue, Hue envía las sentencias de HQL al servidor de Hive a través de las API de Hue Beeswax y muestra los resultados de ejecución en la interfaz de usuario web.

6.13.3 Funciones de código abierto mejoradas de Hue

Funciones de código abierto mejoradas de Hue

- Política de almacenamiento: el número de copias de archivos HDFS varía según el medio de almacenamiento. Esta característica le permite establecer manualmente una política de almacenamiento de directorios HDFS o puede ajustar automáticamente la política de almacenamiento de archivos, modificar el número de copias de archivos, mover el directorio de archivos y eliminar archivos según el tiempo de acceso y el tiempo de modificación más recientes de los archivos HDFS para utilizar plenamente la capacidad de almacenamiento y mejorar el rendimiento del almacenamiento.
- Motor de MR: Puede utilizar el motor MapReduce para ejecutar sentencias SQL de Hive.
- Mejora de la confiabilidad: Hue se despliega en modo activo/en espera. Cuando se interconecta con HDFS, Oozie, Hive, y YARN, Hue puede funcionar en modo de conmutación por error o balanceo de carga.

6.14 Impala

Impala

Impala proporciona consultas de SQL rápidas e interactivas directamente en sus datos de Apache Hadoop almacenados en HDFS, HBase o Object Storage Service (OBS). Además de utilizar la misma plataforma de almacenamiento unificado, Impala también utiliza los mismos metadatos, sintaxis de SQL (Hive SQL), controlador de ODBC e interfaz de usuario (UI de consulta de Impala en Hue) que Apache Hive. Esto proporciona una plataforma familiar y unificada para consultas en tiempo real o por lotes. Impala es una adición a las herramientas disponibles para la consulta de big data. Impala no reemplaza los marcos de procesamiento por lotes construidos en MapReduce como Hive. Hive y otros marcos construidos sobre MapReduce son los más adecuados para trabajos por lotes de larga duración.

Impala proporciona las siguientes características:

- Características más comunes de SQL-92 de Hive Query Language (HiveQL) incluidos SELECT, JOIN y funciones agregadas
- Almacenamiento de HDFS, HBase y OBS, que incluye:
  - Formatos de archivo HDFS: archivos de texto delimitados, Parquet, Avro, SequenceFile y RCFile
  - Códecs de compresión: Snappy, GZIP, Deflate, BZIP
- Interfaces de acceso a datos comunes que incluyen:
  - Controlador de JDBC
  - Controlador de ODBC
  - Hue Beeswax y la UI de consulta de Impala
- Interfaz de línea de comandos de **impala-shell**
- Autenticación de Kerberos

Impala se aplica al análisis fuera de línea (como el análisis de estado de log y clúster) de consultas de datos en tiempo real, minería de datos a gran escala (como el análisis del comportamiento del usuario, el análisis de la región de interés y la visualización de la región) y otros escenarios.

Para obtener más información sobre Impala, visite **https://impala.apache.org/impala-docs.html**.

Impala consta de tres roles: Impala Daemon (Impalad), Impala StateStore y Impala Catalog Service.

Impala Daemon

El componente central de Impala es el demonio Impala, físicamente representado por el proceso **impalad**.

Algunas de las funciones clave que realiza un Impala daemon son:

- Se ejecuta en todos los nodos de datos.
- Lee y escribe en archivos de datos.
- Acepta consultas transmitidas desde el comando **impala-shell**, Hue, JDBC u ODBC.
- Paraleliza las consultas y transmite los resultados intermedios de la consulta de vuelta al coordinador central.
- Invoca un nodo para devolver los resultados de la consulta al cliente.

Los Impala daemons están en comunicación constante con StateStore para confirmar qué daemons están sanos y pueden aceptar nuevos trabajos.

Impala StateStore

El componente de Impala conocido como StateStore comprueba la salud de todos los Impala daemons en un clúster, y continuamente transmite sus hallazgos a cada uno de esos daemons. Está físicamente representado por un proceso de daemon llamado **statestored**. Solo necesita un proceso de este tipo en un host de un clúster. Si un Impala daemon se desconecta debido a una falla de hardware, error de red, problema de software u otra razón, el StateStore informa a todos los demás Impala daemons para que futuras consultas puedan evitar hacer solicitudes al Impala daemon inalcanzable.

Impala Catalog Service

El componente de Impala conocido como el Catalog Service retransmite los cambios de metadatos de las sentencias de Impala SQL a todos los Impala daemons en un clúster. Está físicamente representado por un proceso daemon llamado **catalogd**. Cuando crea una tabla, carga datos, y así sucesivamente a través de Hive, necesita ejecutar REFRESH o INVALIDATE METADATA en un Impala daemon antes de ejecutar una consulta allí. El servicio de catálogo evita la necesidad de emitir sentencias REFRESH e INVALIDATE METADATA cuando los cambios de metadatos son realizados por sentencias emitidas a través de Impala.

6.15 IoTDB

6.15.1 IoTDB Basic Principles

Database for Internet of Things (IoTDB) is a software system that collects, stores, manages, and analyzes IoT time series data. Apache IoTDB uses a lightweight architecture and features high performance and rich functions.

IoTDB sorts time series and stores indexes and chunks, greatly improving the query performance of time series data. IoTDB uses the Raft protocol to ensure data consistency. In time series scenarios, IoTDB pre-computes and stores data to improve analysis performance. Based on the characteristics of time series data, IoTDB provides powerful data encoding and compression capabilities. In addition, its replica mechanism ensures data security. IoTDB is deeply integrated with Apache Hadoop and Flink to meet the requirements of massive data storage, high-speed data reading, and complex data analysis in the industrial IoT field.

NOTA

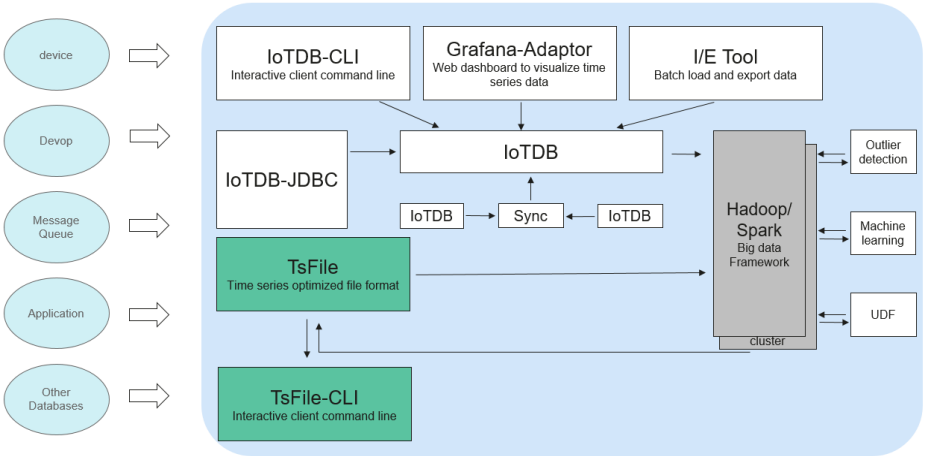
This section applies to MRS 3.1.5 or later.

IoTDB Architecture

The IoTDB suite consists of multiple components to provide a series of functions such as data collection, data writing, data storage, data query, data visualization, and data analysis.

Figura 6-62 shows the overall application architecture after all components of the IoTDB suite are used. IoTDB refers to the time series database component in the suite.

Figura 6-62 IoTDB architecture

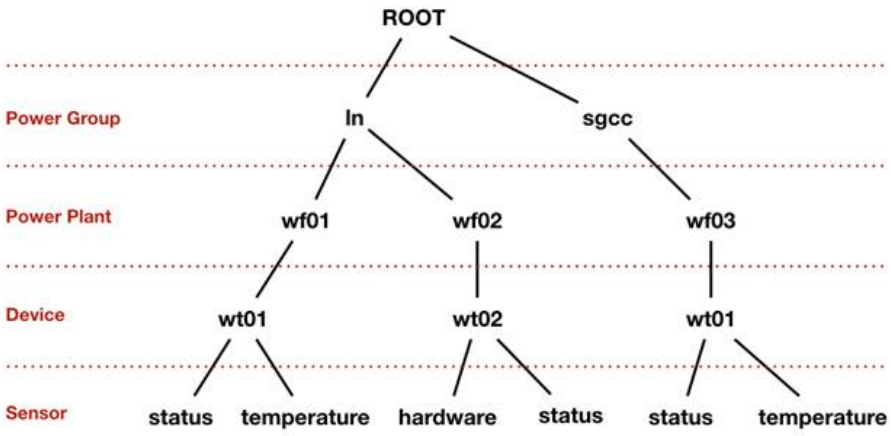


- Users can use Java Database Connectivity (JDBC) to import the time series data and system status data (such as server load, CPU usage and memory usage) collected from device sensors, as well as time series data in message queues, applications, or other databases, to the local or remote IoTDB. Users can also directly write the preceding data into a local TsFile file or a TsFile file in the HDFS.
- Users can write TsFile files to the HDFS to implement data processing tasks such as exception detection and machine learning on the Hadoop or Flink data processing platform.
- The TsFile-Hadoop or TsFile-Flink connector can be used to allow Hadoop or Flink to process the TsFile files written to the HDFS or local host.
- The analysis result can be written back to a TsFile in the same way.
- IoTDB and TsFile also provide client tools to meet users' requirements for viewing and writing data in SQL, script, and graphical formats.

IoTDB Principles

Based on the attribute hierarchy, attribute coverage, and subordinate relationships between data, the IoTDB data model can be represented as the attribute hierarchy, as shown in **Figura 6-63**. The hierarchy is as follows: power group layer - power plant layer - device layer - sensor layer. **ROOT** is a root node, and each node at the sensor layer is a leaf node. According to the IoTDB syntax, the path from **ROOT** to a leaf node is separated by a dot (.). The complete path is used to name a time series in the IoTDB. For example, the time series name corresponding to the path on the left in the following figure is **ROOT.ln.wf01.wt01.status**.

Figura 6-63 IoTDB data model



6.15.2 Relationship Between IoTDB and Other Components

The IoTDB stores data locally, so it does not depend on any other component for storage. However, in a security cluster environment, IoTDB depends on the KrbServer component for Kerberos authentication.

NOTA

This section applies to MRS 3.1.5 or later.

6.15.3 IoTDB Enhanced Open Source Features

This section applies to MRS 3.1.5 or later.

Visualization

- Visualized O&M covers installation, uninstallation, one-click start and stop, configurations, clients, monitoring, alarms, health checks, and logs.
- Visualized permission management does not require background command line operations and supports read and write permission control at the database and table levels.
- Visualized log level configuration dynamically takes effect, supports visualized download and retrieval, and supports log audit.

Security Hardening

User authentication supports Kerberos authentication and SSL encryption, which are compatible with the community authentication mode.

Ecosystem Interconnection

On the basis of native capabilities, the cluster interconnection with MQTT is enhanced.

6.16 Kafka

6.16.1 Principios básicos de Kafka

Kafka es un servicio de registro de confirmación de código abierto, distribuido, particionado y replicado. Kafka es mensajería de publicación y suscripción, repensada como un registro de confirmación distribuido. Proporciona características similares a Java Message Service (JMS) pero otro diseño. Cuenta con resistencia de mensajes, alto rendimiento, métodos distribuidos, soporte para múltiples clientes y tiempo real. Se aplica tanto al consumo de mensajes en línea como fuera de línea, como la recopilación regular de mensajes, el seguimiento de la actividad del sitio web, la agregación de datos estadísticos de operación del sistema (datos de supervisión) y la recopilación de registros. Estos escenarios involucran grandes cantidades de recopilación de datos para servicios de Internet.

Estructura de Kafka

Los productores publican datos sobre temas y los consumidores se suscriben a los temas y consumen mensajes. Un broker es un servidor en un clúster de Kafka. Para cada topic, el clúster de Kafka mantiene particiones para la escalabilidad, el paralelismo y la tolerancia a fallas. Cada partición es una secuencia ordenada e inmutable de mensajes que se agrega continuamente a - un registro de confirmación. A cada mensaje de una partición se le asigna un ID secuencial, que se denomina desplazamiento.

Figura 6-64 Arquitectura de Kafka

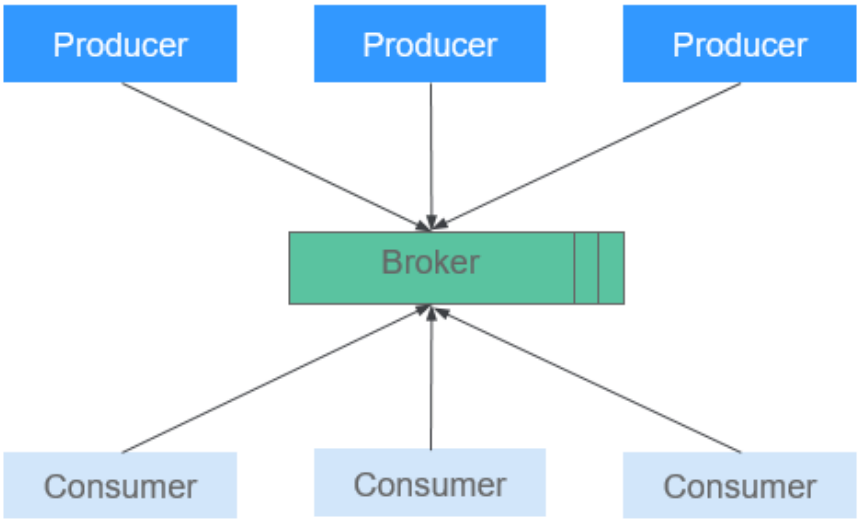
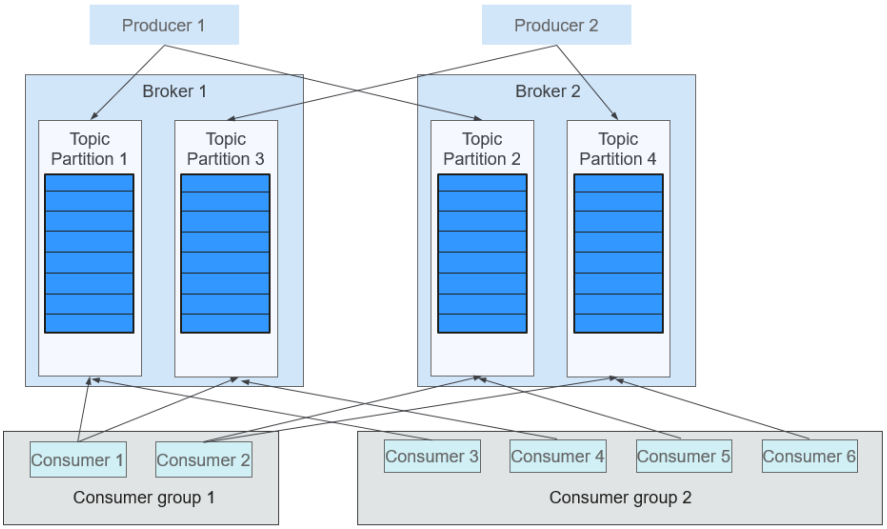


Tabla 6-16 Descripción de la arquitectura de Kafka

Nombre	Descripción
Broker	Un broker es un servidor en un clúster de Kafka.
Topic	Un topic es un nombre de categoría o fuente en el que se publican mensajes. Un topic se puede dividir en varias particiones, que pueden actuar como una unidad paralela.
Partition	Un partition es una secuencia ordenada e inmutable de mensajes que se agrega continuamente a - un registro de confirmación. A cada uno de los mensajes de las particiones se le asigna un número de ID secuencial denominado desplazamiento que identifica de manera única cada mensaje dentro de la partición.
Producer	Los productores publican mensajes a un tema de Kafka.
Consumer	Los consumidores se suscriben a temas y procesan el feed de mensajes publicados.

Figura 6-65 muestra las relaciones entre módulos.

Figura 6-65 Relaciones entre módulos de Kafka



Los consumidores se etiquetan a sí mismos con un nombre de grupo de consumidores, y cada mensaje publicado en un tema se envía a una instancia de consumidor dentro de cada grupo de consumidores que se suscribe. Si todas las instancias de consumidores pertenecen al mismo grupo de consumidores, las cargas se distribuyen uniformemente entre los consumidores. Como se muestra en la figura anterior, el Consumer1 y el Consumer2 trabajan en modo de carga compartida; el Consumer3, Consumer4, Consumer5, and Consumer6 trabajan en modo de carga compartida. Si todas las instancias de consumidor pertenecen a diferentes grupos de consumidores, los mensajes se transmiten a todos los consumidores. Como se muestra en la figura anterior, los mensajes en el Tema 1 se transmiten a todos los consumidores en Consumer Group1 y Consumer Group2.

Para obtener más información sobre la arquitectura y los principios de Kafka, consulte **https://kafka.apache.org/24/documentation.html**.

Principio

- **Confiabilidad de mensaje**

  Cuando un Kafka broker recibe un mensaje, almacena el mensaje en un disco de forma persistente. Cada partición de un tema tiene varias réplicas almacenadas en diferentes nodos de agente. Si un nodo es defectuoso, se pueden utilizar las réplicas de otros nodos.

- **Alto rendimiento**

  Kafka proporciona un alto rendimiento de las siguientes maneras:

  - Los mensajes se escriben en discos en lugar de almacenarse en caché en la memoria, utilizando completamente el rendimiento secuencial de lectura y escritura de discos.
  - El uso de zero-copy elimina las operaciones de E/S.
  - Los datos se envían por lotes, lo que mejora la utilización de la red.
  - Cada tema se divide en varias particiones, lo que aumenta el procesamiento simultáneo. Las operaciones de lectura y escritura simultáneas se pueden realizar entre múltiples productores y consumidores. Los productores envían mensajes a las particiones especificadas en función del algoritmo utilizado.

- **Mecanismo de notificación-suscripción de mensaje**

  Los consumidores se suscriben a temas interesados y consumen datos en modo pull. Los consumidores pueden elegir el modo de consumo, como el consumo de lotes, el consumo repetido y el consumo desde el final, y controlar la velocidad de extracción del mensaje en función de la situación real. Los consumidores necesitan mantener los registros de consumo por sí mismos.

- **Escalabilidad**

  Cuando se agregan nodos de broker para ampliar la capacidad del clúster de Kafka, los brokeres recién agregados se registran con ZooKeeper. Después de que el registro es exitoso, los procedimientos y los consumidores pueden detectar el cambio de manera oportuna y hacer ajustes relacionados.

Características de código abierto

- Confiabilidad

  Se proporcionan métodos de procesamiento de mensajes tales como **At-Least Once**, **At-Most Once** y **Exactly Once**. El estado de procesamiento de mensajes es mantenido por los consumidores. Kafka necesita trabajar con la capa de aplicación para implementar **Exactly Once**.

- Alto rendimiento

  Se proporciona un alto rendimiento para la publicación de mensajes y la suscripción.

- Persistencia

  Los mensajes se almacenan en discos y se pueden usar para el consumo por lotes y programas de aplicación en tiempo real. La persistencia y la replicación de los datos evitan la pérdida de datos.

- Distribución

  Un sistema distribuido es fácil de expandir externamente. Todos los producers, brokers, and consumers apoyan la implementación de múltiples clústeres distribuidos. Los sistemas se pueden escalar sin detener el funcionamiento del software o apagar las máquinas.

Kafka UI

Kafka UI proporciona servicios web de Kafka, muestra información básica sobre módulos funcionales como brokers, topics, partitions, y consumers en un clúster de Kafka, y proporciona entradas de operación para comandos de Kafka comunes. Kafka UI reemplaza a Kafka Manager para proporcionar servicios web seguros de Kafka que cumplen con las especificaciones de seguridad.

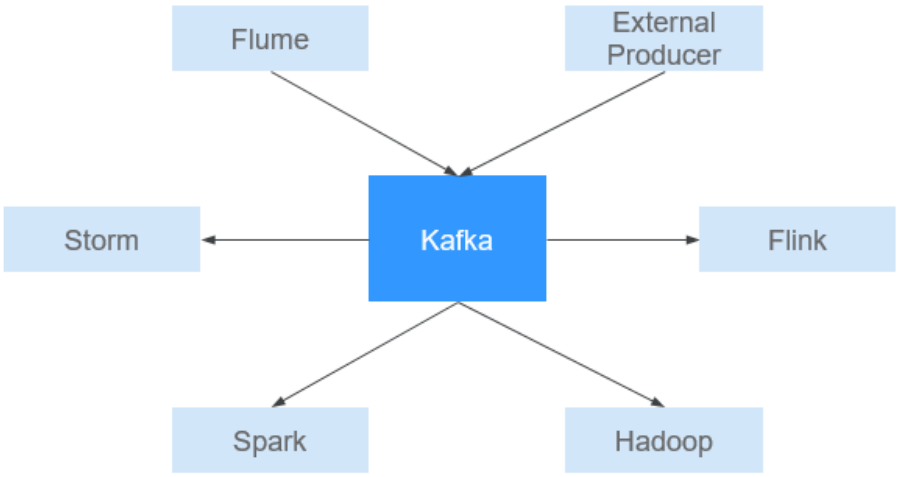
Puede realizar las siguientes operaciones en Kafka UI:

- Comprobar el estado del clúster (topics, consumers, offsets, partitions, replicas, y nodes).
- Redistribuir las particiones en el clúster.
- Crear un topic con configuraciones de topic opcionales.
- Eliminar un tema (compatible cuando **delete.topic.enable** está establecido en **true** para el servicio Kafka).
- Agregar particiones a un tema existente.
- Actualizar las configuraciones de un tema existente.
- Opcionalmente, habilitar el sondeo de JMX para métricas a nivel de broker y a nivel de topic.

6.16.2 Relación entre Kafka y otros componentes

Como un sistema de publicación y suscripción de mensajes, Kafka proporciona métodos de transmisión de datos de alta velocidad para la transmisión de datos entre diferentes subsistemas de la plataforma de FusionInsight. Puede recibir mensajes externos en tiempo real y proporciona los mensajes a los servicios en línea y fuera de línea para su procesamiento. La siguiente figura muestra la relación entre Kafka y otros componentes.

Figura 6-66 Relación con otros componentes



6.16.3 Funciones de código abierto mejoradas de Kafka

Funciones de código abierto mejoradas de Kafka

- Supervisa las siguientes métricas de nivel de topic:
  - Tráfico de entrada de topic
  - Tráfico de salida de topic
  - Tráfico rechazado de topic
  - Número de solicitudes de fetch fallidas por segundo
  - Número de solicitudes de produce fallidas por segundo
  - Número de mensajes de entrada de topic por segundo
  - Número de solicitudes de fetch por segundo
  - Número de solicitudes de produce por segundo
- Consulta la asignación entre los broker ID y las direcciones IP de nodo. En los clientes de Linux, se puede utilizar **kafka-broker-info.sh** para consultar la asignación entre broker ID y las direcciones IP de nodo.

6.17 KafkaManager

KafkaManager es una herramienta para gestionar Apache Kafka y proporciona monitorización y gestión de métricas basadas en GUI de los clústeres de Kafka.

KafkaManager admite las siguientes operaciones:

- Gestionar múltiples clústeres de Kafka.
- Fácil inspección de los estados del clúster (temas, consumidores, desplazamientos, particiones, réplicas y nodos)
- Ejecutar la elección de réplica preferida.
- Generar asignaciones de particiones con la opción de seleccionar brokers para usar.
- Ejecutar reasignación de partición (basada en asignaciones generadas).
- Crear un tema con configuraciones de tema opcionales (se admiten varias versiones de clúster de Kafka).
- Eliminar un tema (solo se admite en 0.8.2+ y **delete.topic.enable=true** se establece en la configuración de broker).
- Generar por lote asignaciones de particiones para múltiples temas con opción de seleccionar brokers para usar.
- Reasignación de particiones de ejecución por lotes para varios temas.
- Agregar particiones a un tema existente.
- Actualizar las configuraciones de un tema existente.
- Opcionalmente, habilitar el sondeo de JMX para métricas a nivel de broker y a nivel de topic.
- Opcionalmente, filtrar los consumidores que no tienen directorios ids/ owner / & offsets/ en el ZooKeeper.

6.18 KrbServer y LdapServer

6.18.1 Principios de KrbServer y LdapServer

Descripción

Para gestionar los permisos de control de acceso en datos y recursos de un clúster, se recomienda que el clúster se instale en modo de seguridad. En modo de seguridad, se debe autenticar una aplicación de cliente y se debe establecer una sesión segura antes de que la aplicación acceda a cualquier recurso del clúster. MRS utiliza KrbServer para proporcionar autenticación de Kerberos para todos los componentes, implementando un mecanismo de autenticación confiable.

LdapServer admite el protocolo ligero de acceso a directorios (LDAP) y proporciona la capacidad de almacenar datos de usuarios y grupos de usuarios para la autenticación de Kerberos.

Arquitectura

La función de autenticación de seguridad para el inicio de sesión del usuario depende de Kerberos y LDAP.

Figura 6-67 Arquitectura de autenticación de seguridad

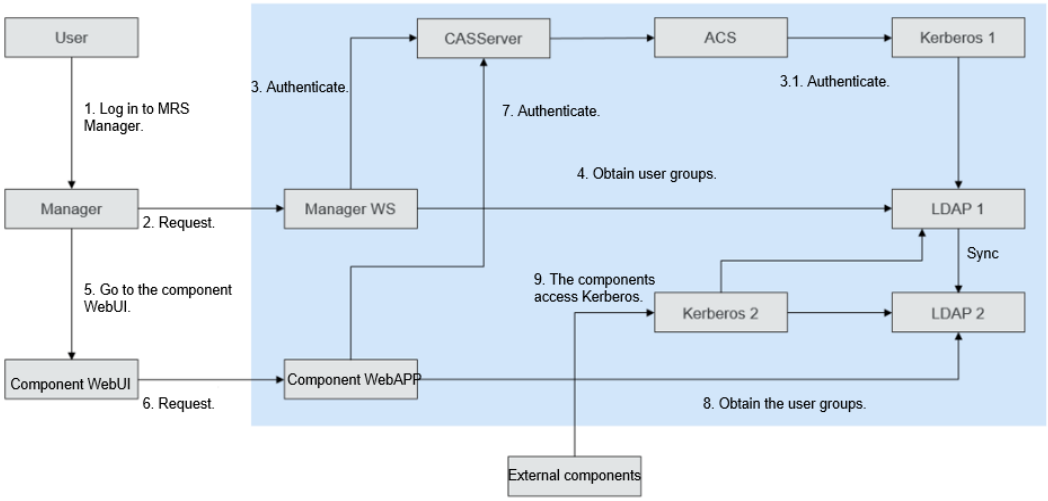


Figura 6-67 incluye tres escenarios:

- Iniciar sesión en la Manager Web UI de MRS

  La arquitectura de autenticación incluye las etapas 1, 2, 3 y 4.
- Iniciar sesión en web UI de componente

  La arquitectura de autenticación incluye las etapas 5, 6, 7 y 8.
- Acceso entre componentes

  La arquitectura de autenticación incluye la etapa 9.

Tabla 6-17 Módulos clave

Nombre de la conexión	Descripción
Manager	Manager de clústeres
Manager WS	WebBrowser
Kerberos1	Servicio de KrbServer (plano de gestión) desplegado en MRS Manager, es decir, OMS Kerberos
Kerberos2	Servicio de KrbServer (plano de servicio) desplegado en el clúster
LDAP1	Servicio LdapServer (plano de gestión) desplegado en MRS Manager, es decir, OMS LDAP
LDAP2	Servicio LdapServer (plano de servicio) desplegado en el clúster

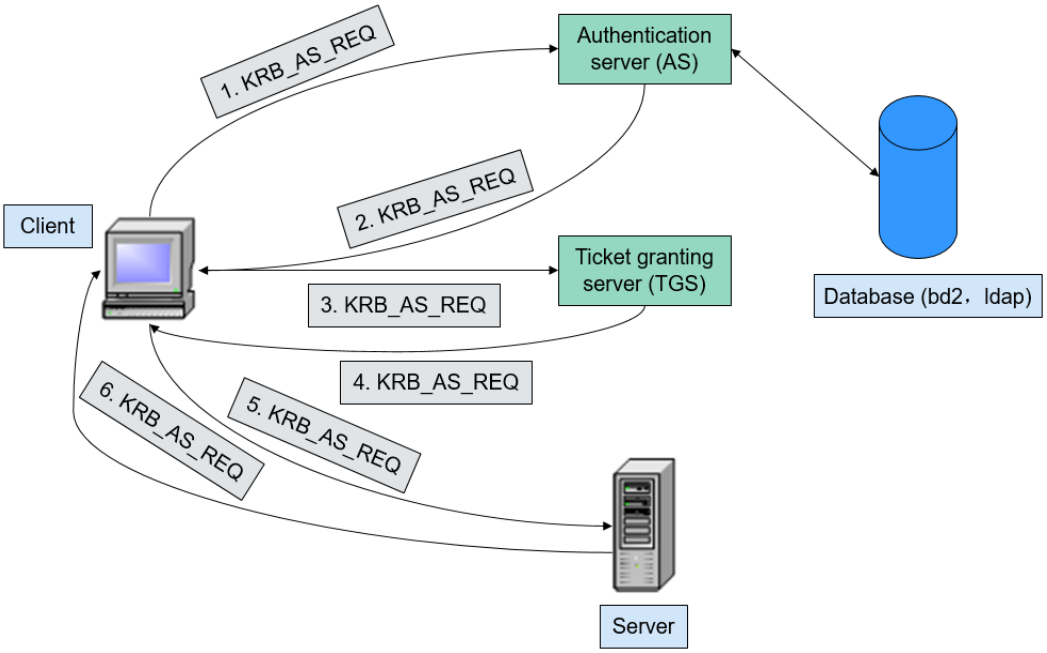
Modo de operación de datos de Kerberos1 en LDAP: Se puede acceder a las instancias activas y en espera de LDAP1 y a las dos instancias en espera de LDAP2 en modo de balanceo de carga. Las operaciones de escritura de datos sólo se pueden realizar en la instancia LDAP1 activa. Las operaciones de lectura de datos se pueden realizar en LDAP1 o LDAP2.

Modo de operación de datos de Kerberos2 en LDAP: Las operaciones de lectura de datos se pueden realizar en LDAP1 y LDAP2. Las operaciones de escritura de datos sólo se pueden realizar en la instancia LDAP1 activa.

Principio

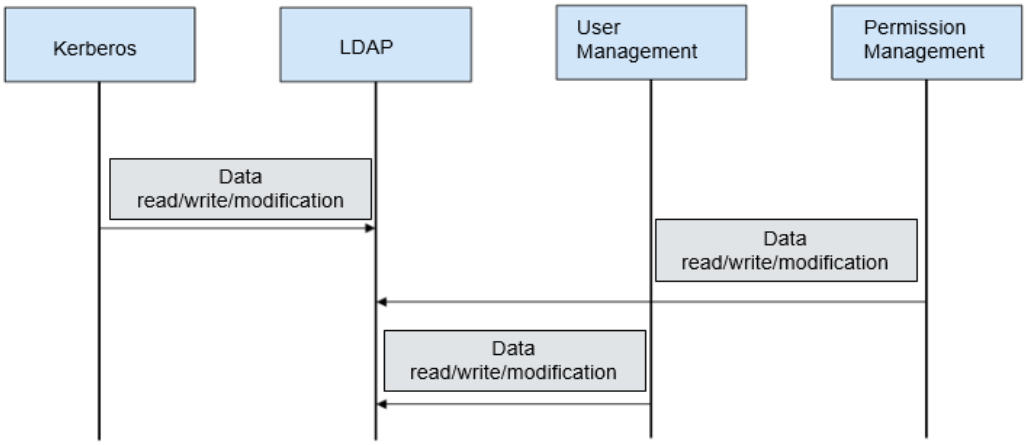
Autenticación de Kerberos

Figura 6-68 Proceso de autenticación



Lectura y escritura de datos de LDAP

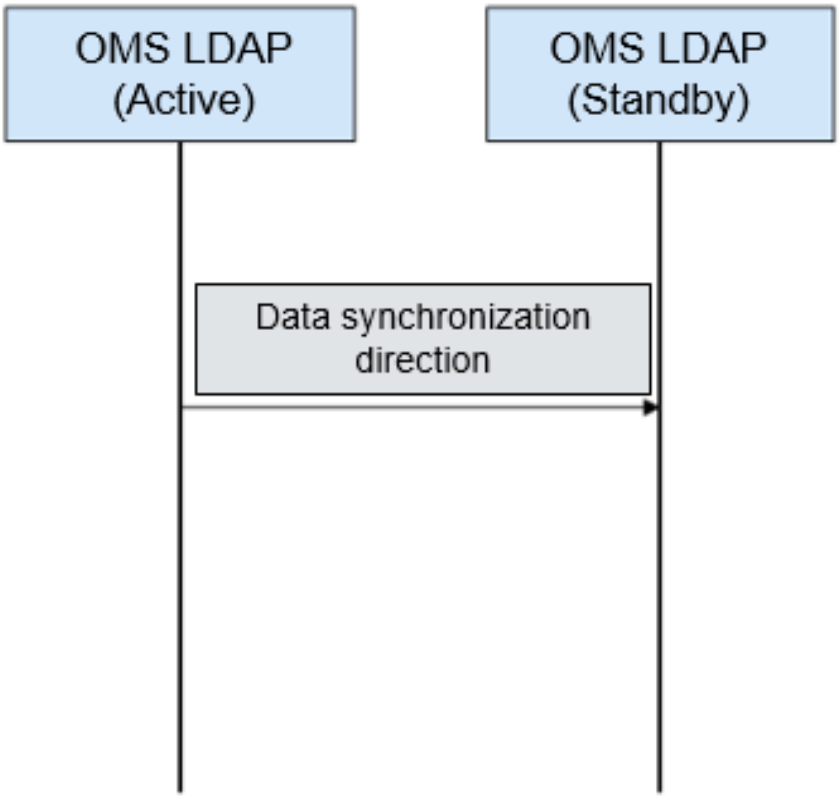
Figura 6-69 Proceso de modificación de datos



Sincronización de datos de LDAP

- Sincronización de datos de LDAP OMS antes de la instalación del clúster

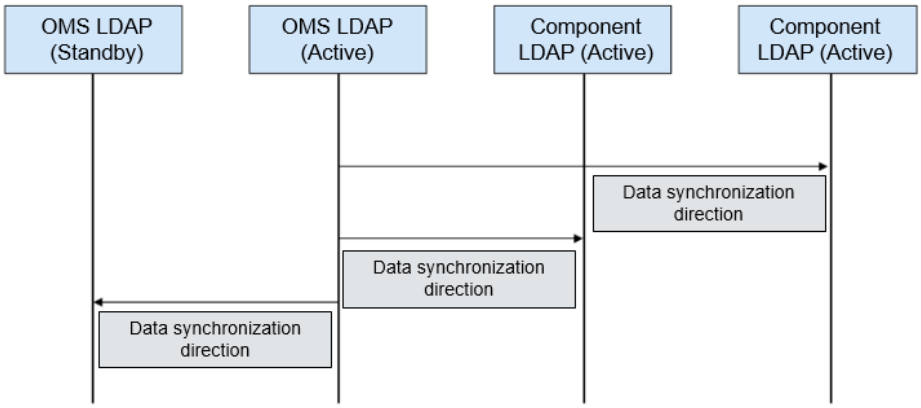
Figura 6-70 Sincronización de datos de OMS LDAP



Dirección de sincronización de datos antes de la instalación del clúster: Los datos se sincronizan desde OMS LDAP activo al OMS LDAP en espera.

- Sincronización de datos de LDAP después de la instalación del clúster

Figura 6-71 Sincronización de datos de LDAP



Dirección de sincronización de datos después de la instalación del clúster: Los datos se sincronizan desde el OMS LDAP activo al OMS LDAP en espera, el componente LDAP en espera y el componente LDAP en espera.

6.18.2 Funciones mejoradas de código abierto de KrbServer y LdapServer

Funciones mejoradas de código abierto de KrbServer y LdapServer: autenticación de servicio dentro del clúster

En un clúster de MRS que utiliza el modo de seguridad, el acceso mutuo entre servicios se implementa basándose en la arquitectura de seguridad de Kerberos. Cuando se va a iniciar un servicio (como HDFS) en el clúster, la clave de sesión correspondiente (keytab, utilizada para la autenticación de identidad de la aplicación) se obtiene de Kerberos. Si otro servicio (como YARN) necesita acceder a HDFS y agregar, eliminar, modificar o consultar datos en HDFS, se deben obtener los TGT y ST correspondientes para un acceso seguro.

Funciones mejoradas de código abierto de KrbServer y LdapServer: Autenticación de desarrollo de aplicaciones

Los componentes MRS proporcionan interfaces de desarrollo de aplicaciones para clientes o clústeres de productos de servicio de capa superior. Durante el desarrollo de aplicaciones, un clúster en modo de seguridad proporciona interfaces de autenticación de desarrollo de aplicaciones especificadas para implementar la autenticación y el acceso de seguridad de aplicaciones. Por ejemplo, la clase UserGroupInformation proporcionada por la API de hadoop-common proporciona varias API de autenticación de seguridad.

- **setConfiguration()** se utiliza para obtener la configuración relacionada y establecer parámetros tales como variables globales.
- **loginUserFromKeytab():** se utiliza para obtener interfaces de TGT.

Funciones mejoradas de código abierto de KrbServer y LdapServer: confianza mutua entre sistemas

MRS proporciona la función de confianza mutua entre dos Managers para implementar operaciones de lectura y escritura de datos entre sistemas.

6.19 Kudu

Kudu es un administrador de almacén de columnas desarrollado para la plataforma Apache Hadoop. Kudu comparte las propiedades técnicas comunes de las aplicaciones del ecosistema de Hadoop, es decir, se ejecuta en hardware básico, que es escalable horizontalmente, ofreciendo alta disponibilidad.

El diseño de Kudu tiene los siguientes beneficios:

- Procesamiento rápido de cargas de trabajo OLAP
- Integración con MapReduce y Spark y otros componentes del ecosistema de Hadoop
- Estrecha integración con Apache Impala, por lo que es una buena alternativa mutable al uso de HDFS con Apache Parquet
- Modelo de consistencia fuerte pero flexible, que le permite elegir los requisitos de consistencia por solicitud, incluida la opción de consistencia estrictamente serializable
- Rendimiento sólido para ejecutar cargas de trabajo secuenciales y aleatorias simultáneamente

- Fácil de gestionar
- Servers y Masters de tabletas de alta disponibilidad utilizan el algoritmo de consenso de Raft, que garantiza que mientras más de la mitad del número total de réplicas esté disponible, la tableta esté disponible para lecturas y escrituras. Por ejemplo, si 2 de cada 3 réplicas o 3 de cada 5 réplicas están disponibles, la tableta está disponible. Las lecturas pueden ser atendidas por tabletas seguidoras de solo lectura, incluso en caso de falla de una tableta líder.
- Modelo de datos estructurados

Al combinar todas estas propiedades, Kudu se dirige a la compatibilidad con familias de aplicaciones que son difíciles o imposibles de implementar en las tecnologías de almacenamiento de Hadoop de generación actual.

Algunos ejemplos de aplicaciones para las que Kudu es una gran solución son:

- Aplicaciones de informes en las que los datos recién llegados deben estar disponibles de inmediato para los usuarios finales
- Aplicaciones de serie temporal que deben soportar simultáneamente consultas a través de grandes cantidades de datos históricos y consultas granulares sobre una entidad individual que debe regresar muy rápidamente
- Aplicaciones que utilizan modelos predictivos para tomar decisiones en tiempo real con actualizaciones periódicas del modelo predictivo basadas en todos los datos históricos

6.20 Loader

6.20.1 Principios básicos del Loader

Loader está desarrollado basado en el componente Sqoop de código abierto. Se utiliza para intercambiar datos y archivos entre MRS y bases de datos relacionales y sistemas de archivos. Loader puede importar datos desde bases de datos relacionales o servidores de archivos a los componentes HDFS y HBase, o exportar datos desde HDFS y HBase a bases de datos relacionales o servidores de archivos.

Un modelo de cargador consta de cliente de cargador y servidor de carga, como se muestra en **Figura 6-72**.

Figura 6-72 Modelo de Loader

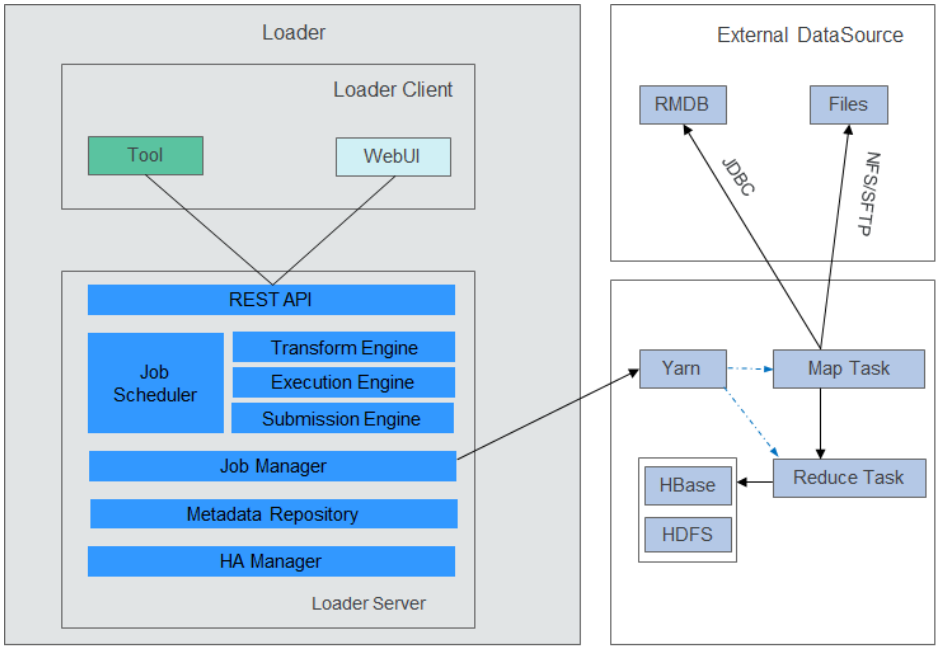


Tabla 6-18 describe las funciones de cada módulo mostrado en la figura anterior.

Tabla 6-18 Componentes del modelo de Loader

Módulo	Descripción
Loader Client	Cliente de Loader. Proporciona dos interfaces: web UI and CLI.
Loader Server	Servidor de Loader. Procesa solicitudes de operación enviadas desde el cliente, gestiona conectores y metadatos, envía trabajos de MapReduce y supervisa el estado de los trabajos de MapReduce.
REST API	Proporciona una API de Transferencia de Estado Representacional (RESTful) (HTTP + JSON) para procesar las solicitudes de operación enviadas desde el cliente.
Job Scheduler	Programador de trabajo simple. Ejecuta periódicamente trabajos de Loader.
Transform Engine	Motor de transformación de datos. Es compatible con la combinación de campo, el corte de cadena y el reverso de cadena.
Execution Engine	Motor de ejecución de trabajos de Loader. Ejecuta trabajos de Loader de manera de MapReduce.
Submission Engine	Motor de envío de trabajos de Loader. Envía trabajos de Loader a MapReduce.
Job Manager	Gestiona trabajos de Loader, incluyidas la creación, consulta, actualización, eliminación, activación, desactivación, inicio y detención de trabajos.

Módulo	Descripción
Metadata Repository	Repositorio de metadatos. Almacena y gestiona datos sobre conectores de Loader, procedimientos de transformación y trabajos.
HA Manager	Gestiona el estado activo/en espera de los procesos de Loader Server. El Loader Server tiene dos nodos que se implementan en modo activo/en espera.

Loader importa o exporta trabajos en paralelo con trabajos de MapReduce. Algunos trabajos de importación o exportación pueden implicar solo las operaciones de Map, mientras que otros pueden implicar tanto las operaciones de Map como las de Reduce.

Loader implementa la tolerancia a fallas mediante MapReduce. Los trabajos se pueden reprogramar cuando se produce un error en la ejecución de un trabajo.

- **Importación a datos a HBase**

  Cuando se realiza la operación de Map para trabajos de MapReduce, Loader obtiene datos de un origen de datos externo.

  Cuando se realiza una operación Reduce para un trabajo de MapReduce, Loader habilita el mismo número de tareas de Reduce según el número de Regions. Las tareas Reduce reciben datos de tareas de Map, generan HFiles por Region y almacenan los HFiles en un directorio temporal de HDFS.

  Cuando se envía un trabajo de MapReduce, Loader migra HFiles desde el directorio temporal al directorio HBase.

- **Importación de datos a HDFS**

  Cuando se realiza una operación de Map para un trabajo de MapReduce, Loader obtiene datos de un origen de datos externo y los exporta a un directorio temporal (denominado *export directory*-**ldtmp**).

  Cuando se envía un trabajo de MapReduce, Loader migra los datos del directorio temporal al directorio de salida.

- **Exportación de datos a una base de datos relacional**

  Cuando se realiza una operación de Map para un trabajo de MapReduce, Loader obtiene datos de HDFS o HBase e inserta los datos en una tabla temporal (Staging Table) a través de la API de conectividad DataBase de Java (JDBC).

  Cuando se envía un trabajo de MapReduce, Loader migra datos de la tabla temporal a una tabla formal.

- **Exportación de datos a un sistema de archivo**

  Cuando se realiza una operación de Map para un trabajo de MapReduce, Loader obtiene datos de HDFS o HBase y escribe los datos en un directorio temporal del servidor de archivos.

  Cuando se envía un trabajo de MapReduce, Loader migra datos del directorio temporal a un directorio formal.

Para obtener más información sobre la arquitectura y los principios del cargador, consulte **https://sqoop.apache.org/docs/1.99.3/index.html**.

6.20.2 Relación entre el cargador y otros componentes

Los componentes que interactúan con Loader incluyen HDFS, HBase, MapReduce y ZooKeeper. Loader funciona como cliente para usar ciertas funciones de estos componentes, como almacenar datos en HDFS y HBase y leer datos de tablas HDFS y HBase. Además, Loader funciona como un cliente de MapReduce para importar o exportar datos.

6.20.3 Funciones de código abierto mejoradas de Loader

Función de código abierto mejorado de Loader: importación y exportación de datos

Loader está desarrollado basado en Sqoop. Además de las funciones de Sqoop, Loader tiene las siguientes características mejoradas:

- Proporciona funciones de conversión de datos.
- Soporta conversión de configuración basada en GUI.
- Importa datos de un servidor SFTP/FTP a HDFS/OBS.
- Importa datos de un servidor SFTP/FTP a una tabla HBase.
- Importa datos de un servidor SFTP/FTP a una tabla Phoenix.
- Importa datos de un servidor SFTP/FTP a una tabla Hive.
- Exporta datos de HDFS/OBS a un servidor SFTP/FTP.
- Exporta datos de una tabla HBase a un servidor SFTP/FTP.
- Exporta datos de una tabla Phoenix a un servidor SFTP/FTP.
- Importa datos de una base de datos relacional a una tabla HBase.
- Importa datos de una base de datos relacional a una tabla de Phoenix.
- Importa datos de una base de datos relacional a una tabla Hive.
- Exporta datos de una tabla HBase a una base de datos relacional.
- Exporta datos de una tabla Phoenix a una base de datos relacional.
- Importa datos de una tabla particionada de Oracle a HDFS/OBS.
- Importa datos de una tabla particionada de Oracle a una tabla HBase.
- Importa datos de una tabla particionada de Oracle a una tabla de Phoenix.
- Importa datos de una tabla particionada de Oracle a una tabla Hive.
- Exporta datos de HDFS/OBS a una tabla particionada de Oracle.
- Exporta datos de HBase a una tabla particionada de Oracle.
- Exporta datos de una tabla Phoenix a una tabla particionada de Oracle.
- Importa datos de HDFS a una tabla HBase, una tabla Phoenix y una tabla Hive en el mismo clúster.
- Exporta datos de una tabla HBase y una tabla Phoenix a HDFS/OBS en el mismo clúster.
- Importa datos a una tabla HBase y a una tabla Phoenix mediante **bulkload** o **put list**.
- Importa todo tipo de archivos desde un servidor de SFTP/FTP a HDFS. El componente de código abierto Sqoop solo puede importar archivos de texto.
- Exporta todo tipo de archivos desde HDFS/OBS a un servidor SFTP. El componente de código abierto Sqoop puede exportar solo archivos de texto y archivos SequenceFile.

- Soporta la conversión de formato de codificación de archivos durante la importación y exportación de archivos. Los formatos de codificación compatibles incluyen todos los formatos compatibles con Java Development Kit (JDK).
- Conserva la estructura original de directorios y los nombres de archivo durante la importación y exportación de archivos.
- Admite la combinación de archivos durante la importación y exportación de archivos. Por ejemplo, si se va a importar un gran número de archivos, estos archivos se pueden combinar en archivos *n* (puede configurarse el *n*).
- Soporta el filtrado de archivos durante la importación y exportación de archivos. Las reglas de filtrado admiten comodines y expresiones regulares.
- Admite la importación y exportación por lotes de tareas ETL.
- Soporta consulta por página y palabra clave y gestión de grupo de tareas ETL.
- Proporciona direcciones IP flotantes para componentes externos.

6.21 Manager

6.21.1 Principios Básicos de Manager

Descripción

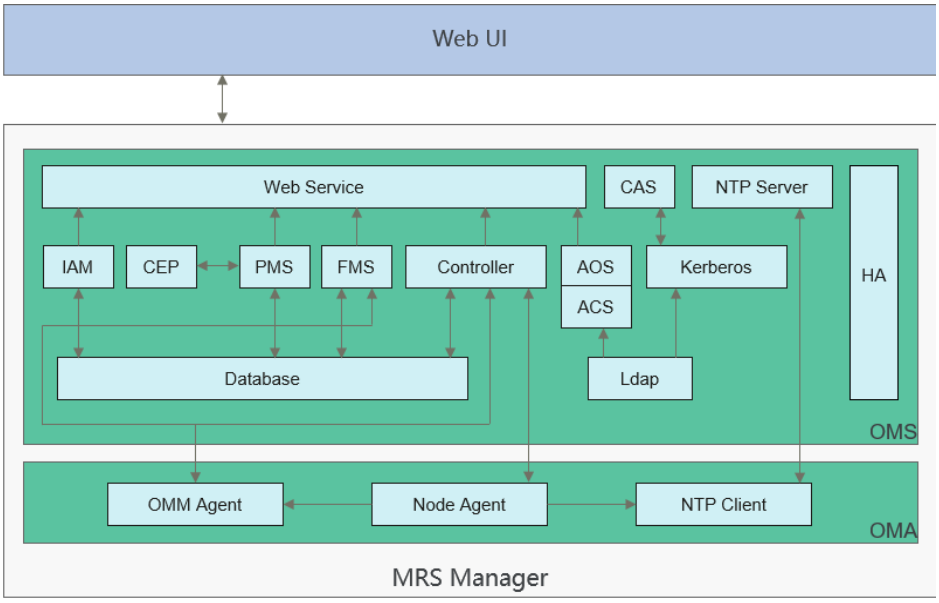
Manager es el sistema de gestión O&M de MRS y proporciona capacidades unificadas de gestión de clústeres para servicios desplegados en clústeres.

Manager proporciona funciones como supervisión del rendimiento, alarmas, gestión de usuarios, gestión de permisos, auditoría, gestión de servicios, comprobación de estado, y recopilación de registros.

Arquitectura

Figura 6-73 muestra la arquitectura lógica general de FusionInsight Manager.

Figura 6-73 Arquitectura lógica de Manager



Manager está formado por OMS y OMA.

- OMS: sirve como nodo de gestión en el sistema O&M. Hay dos nodos OMS desplegados en modo activo/en espera.
- OMA: nodo gestionado en el sistema O&M. Generalmente, hay múltiples nodos OMA.

Tabla 6-19 describe los módulos mostrados en el documento **Figura 6-73**.

Tabla 6-19 Descripción del módulo de servicio

Módulo	Descripción
Web Service	Un servicio web desplegado bajo Tomcat, que proporciona HTTPS API de Manager. Se utiliza para acceder al Manager a través del navegador web. Además, proporciona la capacidad de acceso en dirección norte basada en los protocolos Syslog y SNMP.
OMS	Nodo de gestión del sistema O&M. Generalmente, hay dos nodos OMS que funcionan en modo activo/en espera.
OMA	Nodo gestionado en el sistema O&M. Generalmente, hay múltiples nodos OMA.
Controller	El centro de control de Manager. Puede converger información de todos los nodos del clúster y mostrarla a los administradores del clúster de MRS, así como recibir de los administradores del clúster de MRS y sincronizar información con todos los nodos del clúster de acuerdo con el rango de instrucciones de operación. Proceso de control de Manager. Implementa diversas acciones de gestión: 1. El servicio web ofrece diversas acciones de gestión (como la instalación, el inicio y la parada del servicio y la modificación de la configuración) al Controller. 2. Controller descompone el comando y entrega la acción a cada Node Agent, por ejemplo, iniciar un servicio implica múltiples roles e instancias. 3. Controller es responsable de supervisar la implementación de cada acción.
Node Agent	Node Agent existe en cada nodo del clúster y es un habilitador de Manager en un solo nodo. ● Node Agent representa todos los componentes desplegados en el nodo para interactuar con Controller, implementando la convergencia de varios nodos de un clúster a un solo nodo. ● Node Agent permite a Controller realizar todas las operaciones en los componentes desplegados en el nodo. Permite implementar funciones de Controller. Node Agent envía mensajes de latidos al Controller en un intervalo de 3 segundos. No se puede configurar el intervalo.
IAM	Registra registros de auditoría. Cada operación sin consulta en la interfaz de usuario de Manager tiene un registro de auditoría relacionado.

Módulo	Descripción
PMS	El módulo de supervisión del rendimiento. Recoge los datos de supervisión de rendimiento en cada OMA y proporciona la función de consulta.
CEP	Módulo de función de convergencia. Por ejemplo, el espacio de disco usado de todos los OMA se recopila como un indicador de rendimiento.
FMS	Módulo de alarma. Recoge y consulta alarmas en cada OMA.
OMM Agent	Agent para monitoreo de rendimiento y reporte de alarmas en el OMA. Recopila datos de supervisión del rendimiento y datos de alarma en Agente Node.
CAS	Centro de autenticación unificado. Cuando un usuario inicia sesión en el servicio web, CAS autentica el inicio de sesión. El navegador redirige automáticamente al usuario al CAS a través de URLs.
AOS	Módulo de gestión de permisos. Gestiona los permisos de usuarios y grupos de usuarios.
ACS	Módulo de gestión de usuarios y grupos de usuarios. Gestiona usuarios y grupos de usuarios a los que pertenecen.
Kerberos	LDAP se depliega en OMS y en un clúster, respectivamente. ● OMS Kerberos proporciona el inicio de sesión único (SSO) y la autenticación entre Controller y Node Agent. ● Kerberos en el clúster proporciona la función de autenticación de seguridad de usuario para los componentes. El nombre del servicio es **KrbServer**, que contiene dos instancias de rol: – KerberosServer: KerberosServer es un servidor de autenticación que proporciona autenticación de seguridad para MRS. – KerberosAdmin: gestiona los procesos de los usuarios de Kerberos.
Ldap	LDAP se depliega en OMS y en un clúster, respectivamente. ● OMS LDAP proporciona almacenamiento de datos para la autenticación del usuario. ● El LDAP en el clúster funciona como la copia de respaldo del OMS LDAP. El nombre del servicio es **LdapServer** y la instancia de rol es **SlapdServer**.
Database	Base de datos de Manager utilizada para almacenar registros y alarmas.
HA	Módulo de gestión de HA que gestiona los OMSs activos y en espera.
NTP Server NTP Client	Sincroniza el reloj del sistema de cada nodo del clúster.

6.21.2 Características clave de Manager

Característica clave: Monitoreo de alarma unificado

Manager proporciona la función de monitorización de alarmas visualizada y conveniente. Los usuarios pueden obtener rápidamente indicadores clave de rendimiento del clúster, evaluar el estado del clúster, personalizar la visualización de indicadores de rendimiento y convertir indicadores en alarmas. Manager puede supervisar el estado de funcionamiento de todos los componentes e informar de las alarmas en tiempo real cuando se producen fallas. La ayuda en línea en la GUI le permite ver los contadores de rendimiento y los métodos de liquidación de alarmas para rectificar rápidamente las fallas.

Característica clave: Gestión de permisos de usuario unificado

Manager proporciona la gestión de permisos de los componentes de una manera unificada.

Manager introduce el concepto de rol y utiliza el control de acceso basado en roles (RBAC) para gestionar los permisos del sistema. Muestra y gestiona de forma centralizada las funciones de permisos dispersos de cada componente en el sistema y organiza los permisos de cada componente en forma de conjuntos de permisos (roles) para formar un concepto de permisos de sistema unificado. Al hacerlo, los usuarios comunes no pueden obtener detalles internos de gestión de permisos, y los permisos se vuelven fáciles de gestionar para los administradores de clústeres de MRS, lo que facilita enormemente la gestión de permisos y mejora la experiencia del usuario.

Característica clave: SSO

El inicio de sesión único (SSO) se proporciona entre la interfaz de usuario web de Manager y la interfaz de usuario web de componentes, así como para la integración entre MRS y sistemas de terceros.

Esta función gestiona y autentica de forma centralizada los usuarios de Manager y los usuarios de componentes. Todo el sistema utiliza LDAP para gestionar usuarios y utiliza Kerberos para la autenticación. Se utiliza un conjunto de mecanismos de gestión de Kerberos y LDAP entre el OMS y los componentes. SSO (incluido el inicio de sesión único y el cierre de sesión único) se implementa a través de CAS. Con el inicio de sesión único, los usuarios pueden cambiar fácilmente tareas entre la interfaz de usuario web de Manager, las interfaces de usuario web de componentes y sistemas de terceros, sin cambiar a otro usuario.

NOTA

- Para garantizar la seguridad, el servidor CAS puede retener un ticket de concesión de tickets (TGT) utilizado por un usuario solo durante 20 minutos.
- Si un usuario no realiza ninguna operación en la página (incluida la interfaz de usuario web de Manager y las interfaces de usuario web de componentes) en 20 minutos, la página se bloquea automáticamente.

Característica clave: Chequeo de salud e inspección automáticos

Manager proporciona a los usuarios una inspección automática de los entornos en funcionamiento del sistema y ayuda a los usuarios a comprobar y auditar el estado de funcionamiento del sistema con un solo clic, asegurando el correcto funcionamiento del sistema y reduciendo los costos de operación y mantenimiento del sistema. Después de ver los resultados de la inspección, puede exportar informes para archivarlos y analizar fallas.

Característica clave: Gestión de tenant

Manager presenta el concepto multitenant. Los recursos de CPU, memoria y disco de un clúster se pueden integrar en un conjunto. El conjunto se llama tenant. Un modo que involucra diferentes tenants se llama modo multitenant.

Manager proporciona la función multitenant, soporta un modelo de tenant basado en niveles y permite que tenants se agreguen y eliminen dinámicamente, logrando el aislamiento de recursos. Como resultado, puede gestionar y configurar dinámicamente los recursos informáticos y los recursos de almacenamiento de tenants.

- Los recursos informáticos indican los recursos de cola de tareas de Yarn de tenants. La cuota de cola de tareas se puede modificar y se pueden ver el estado de uso y las estadísticas de la cola de tareas.
- Los recursos de almacenamiento se pueden almacenar en HDFS. Puede agregar y eliminar los directorios de almacenamiento HDFS de tenants y establecer las cuotas de cantidad de archivos y el espacio de almacenamiento de los directorios.

Como plataforma unificada de gestión de tenants de MRS, MRS Manager permite a los usuarios crear y gestionar tenants en clústeres según los requisitos de servicio.

- Los roles, los recursos informáticos y los recursos de almacenamiento se crean automáticamente cuando se crean los tenants. De forma predeterminada, todos los permisos de los nuevos recursos informáticos y de almacenamiento se asignan a los roles de un tenant.
- Después de modificar los recursos informáticos o de almacenamiento de tenant, los permisos de las funciones de tenant se actualizan automáticamente.

Manager también proporciona la función multiinstancia para que los usuarios puedan utilizar HBase, Hive o Spark solos en el escenario de control de recursos y aislamiento de servicios. La función multiinstancia está deshabilitada de forma predeterminada y se puede activar manualmente.

Característica clave: Soporte multilingüe

Manager admite varios idiomas y selecciona automáticamente chino o inglés según la preferencia de idioma del navegador. Si el idioma preferido del navegador es chino, Manager mostrará el portal en chino; si el idioma preferido del navegador no es chino, Manager mostrará el portal en inglés. También puede cambiar entre chino e inglés en la esquina inferior izquierda de la página según su preferencia de idioma. (Solo MRS 3.x y versiones posteriores admiten el cambio con un solo clic entre chino e inglés.)

6.22 MapReduce

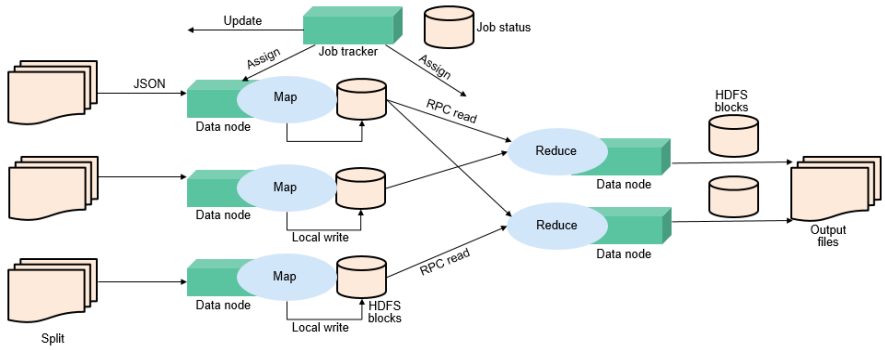
6.22.1 Principios básicos de MapReduce

MapReduce es el núcleo de Hadoop. Como arquitectura de software propuesta por Google, MapReduce se utiliza para el cálculo paralelo de conjuntos de datos a gran escala (más de 1 TB). Los conceptos "Map" y "Reduce" y sus pensamientos principales son tomados del lenguaje de programación funcional y también tomados de las características del lenguaje de programación vectorial.

La implementación de software actual es la siguiente: Especificar una función de Map para mapear una serie de pares de clave-valor en una nueva serie de pares de clave-valor, y

especificar una función Reduce para asegurarse de que todos los valores en los pares clave-valor mapeados comparten la misma clave.

Figura 6-74 Motor de procesamiento por lotes distribuido



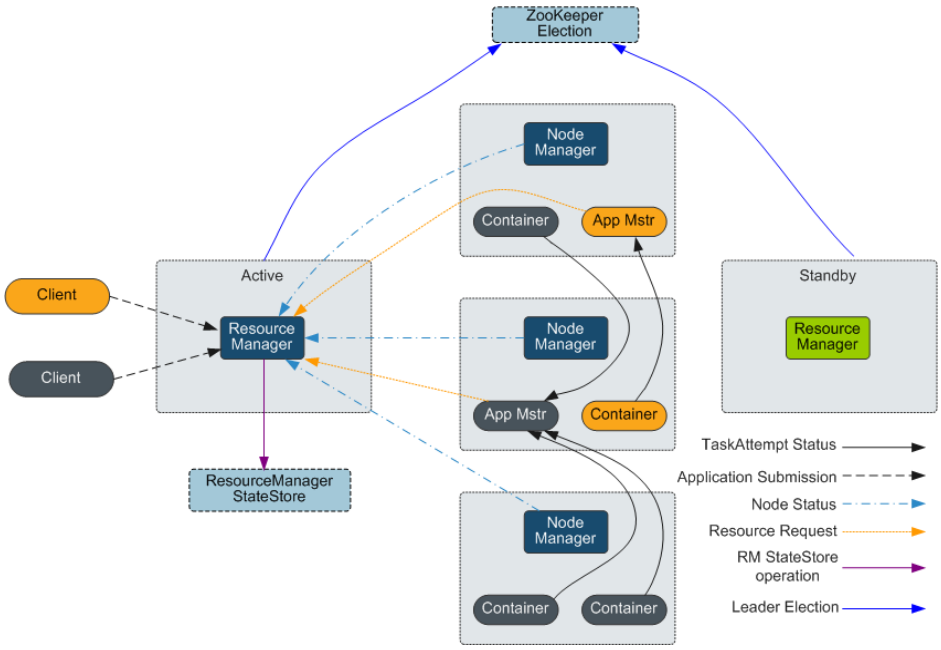
MapReduce es un marco de software para procesar grandes conjuntos de datos en paralelo. La raíz de MapReduce es las funciones de Map y Reduce en la programación funcional. La función Map acepta un grupo de datos y los transforma en una lista de pares de clave-valor. Cada elemento en el dominio de entrada corresponde a un par de clave-valor. La función Reduce acepta la lista generada por la función Map y, a continuación, reduce la lista de pares de clave-valor en función de las claves. MapReduce divide una tarea en varias partes y las asigna a diferentes dispositivos para su procesamiento. De esta manera, la tarea puede finalizarse en un entorno distribuido en lugar de en un único servidor potente.

Para obtener más información, consulte **Tutorial de MapReduce**.

Estructura de MapReduce

Como se muestra en **Figura 6-75**, MapReduce se integra en YARN a través de las interfaces de Cliente y ApplicationMaster de YARN, y utiliza YARN para solicitar recursos informáticos.

Figura 6-75 Arquitectura básica de Apache YARN y MapReduce



6.22.2 Relación entre MapReduce y otros componentes

Relación entre MapReduce y HDFS

- HDFS cuenta con alta tolerancia a fallas y alto rendimiento, y se puede implementar en hardware de bajo costo para almacenar datos de aplicaciones con conjuntos de datos masivos.
- MapReduce es un modelo de programación utilizado para el cálculo paralelo de grandes conjuntos de datos (mayores de 1 TB). Los datos computados por MapReduce provienen de múltiples orígenes de datos, como FileSystem locales, HDFS y bases de datos. La mayoría de los datos provienen del HDFS. El alto rendimiento de HDFS se puede utilizar para leer datos masivos. Después de ser computados, los datos se pueden almacenar en HDFS.

Relación entre el MapReduce y Yarn

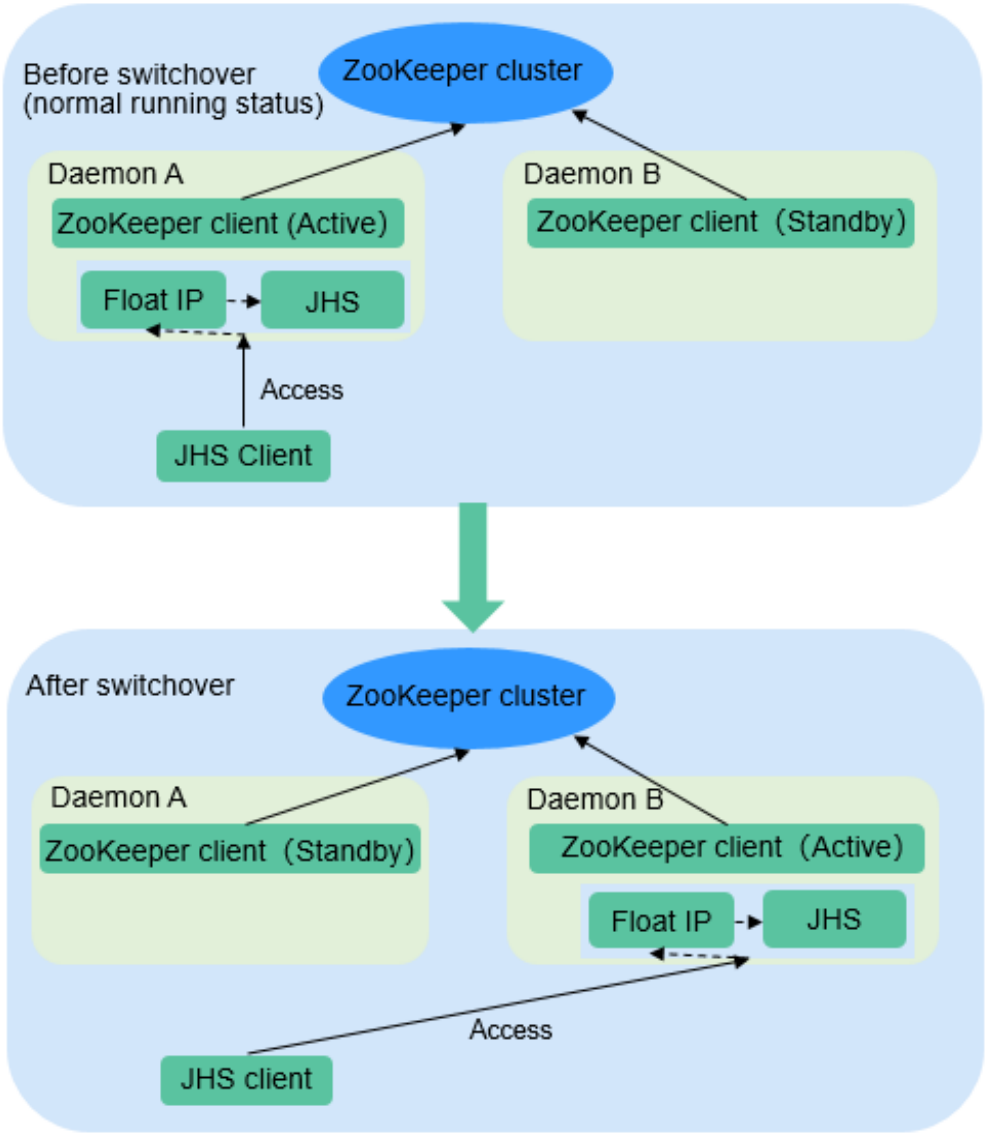
MapReduce es un marco de computación que se ejecuta en Yarn, que se utiliza para el procesamiento por lotes. MRv1 se implementa basado en MapReduce en Hadoop 1.0, que se compone de modelos de programación (Las API de programación nuevas y antiguas), entorno de ejecución (JobTracker y TaskTracker), y motor de procesamiento de datos (MapTask y ReduceTask). Este marco sigue siendo débil en escalabilidad, tolerancia a fallas (JobTracker SPOF) y compatibilidad con varios marcos. (Actualmente, solo se admite el marco de computación de MapReduce.) MRv2 se implementa en base a MapReduce en Hadoop 2.0. El código fuente reutiliza los modelos de programación MRv1 y la implementación del motor de procesamiento de datos, y el entorno de ejecución se compone de ResourceManager y ApplicationMaster. ResourceManager es un nuevo sistema de gestión de recursos, y ApplicationMaster es responsable de cortar los datos de trabajo de MapReduce, asignar tareas, solicitar recursos, programar tareas y tolerar fallas.

6.22.3 Funciones mejoradas de código abierto de MapReduce

Función mejorada de código abierto de MapReduce: JobHistoryServer HA

JobHistoryServer (JHS) es el servidor utilizado para ver la información de tareas de MapReduce históricas. Actualmente, el JHS de código abierto solo admite servicios de instancia única. JHS HA puede resolver el problema de que una aplicación no puede acceder a la API MapReduce cuando se producen SPOFs en el JHS, lo que hace que la aplicación no se ejecute. Esto mejora considerablemente la alta disponibilidad del servicio MapReduce.

Figura 6-76 Transición de estado de la conmutación activa/en espera de JobHistoryServer HA



Alta disponibilidad de JobHistoryServer

- ZooKeeper se utiliza para implementar la elección activa/en espera y la conmutación.

- JHS utiliza la dirección IP flotante para proporcionar servicios externamente.
- Se admiten tanto los modos de implementación de instancia única de JHS como los modos de implementación de HA.
- Solo un nodo inicia el proceso JHS en un punto de tiempo para evitar que varias operaciones JHS procesen el mismo archivo.
- Puede realizar escalamiento horizontal, encogimiento, migración de instancia, actualización y comprobación de estado.

Función mejorada de código abierto: mejora del rendimiento de MapReduce mediante optimización del proceso de fusión/ordenación en escenarios específicos

La siguiente figura muestra el flujo de trabajo de una tarea de MapReduce.

Figura 6-77 MapReduce job

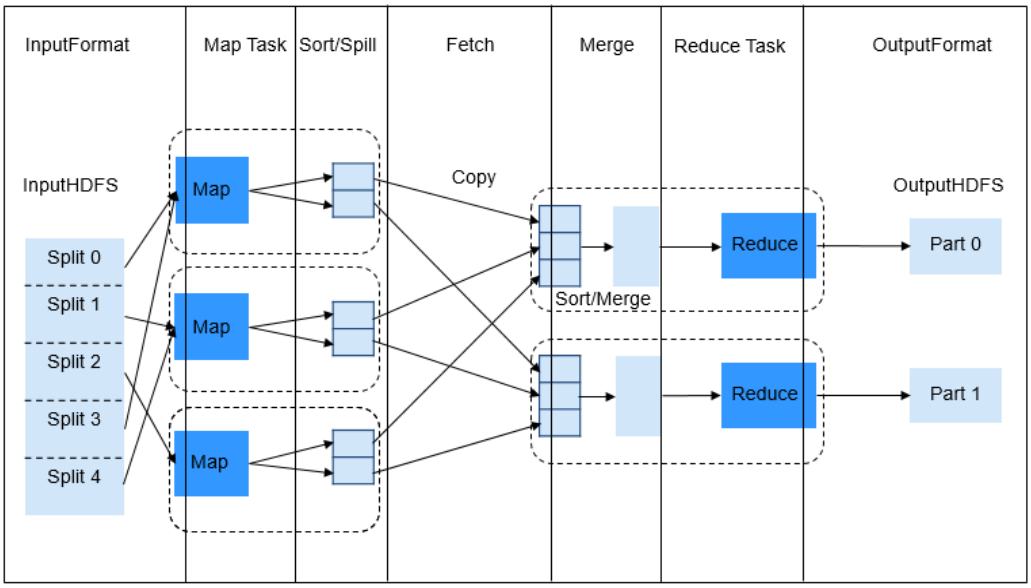
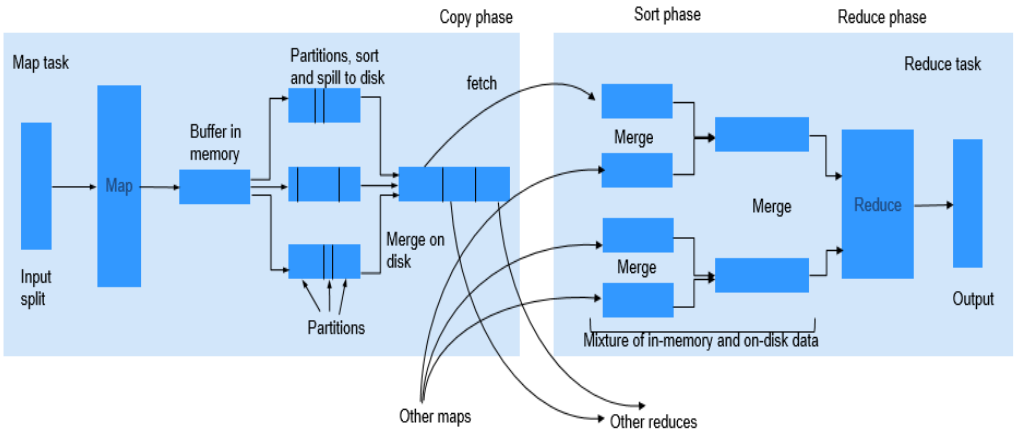


Figura 6-78 Flujo de ejecución de job de MapReduce



El proceso Reducir se divide en tres pasos diferentes: Copy, Sort (en realidad se supone que se llama Merge) y Reduce. En la fase de Copy, Reducer intenta obtener la salida de Maps de NodeManagers y almacenarla en Reducer ya sea en memoria o en disco. A continuación, comienza la fase de Shuffle (Sort and Merge). Todas las salidas de mapa obtenidas se están ordenando, y los segmentos de diferentes salidas de mapa se fusionan antes de enviarse a Reducer. Cuando un job tiene un gran número de maps que se procesarán, el proceso de shuffle consume mucho tiempo. Para tareas específicas (por ejemplo, tareas de SQL como hash join y hash aggregation), la ordenación no es obligatoria durante el proceso de shuffle. Sin embargo, la ordenación se requiere de forma predeterminada en el proceso de shuffle.

Esta función se mejora mediante el uso de la API MapReduce, que puede cerrar automáticamente el proceso Sort para dichas tareas. Cuando la clasificación está deshabilitada, la API fusiona directamente los datos de salida de Maps obtenidos y los envía a Reducer. Esto ahorra mucho tiempo y mejora significativamente la eficiencia de las tareas de SQL.

Función mejorada de código abierto: Problema de archivo de log pequeño resuelto después de la optimización de MR History Server

Después de ejecutar el job que se ejecuta en Yarn, NodeManager utiliza LogAggregationService para recopilar y enviar logs generados a HDFS y los elimina del sistema de archivos local. Una vez que logs se almacenan en HDFS, son gestionados por MR HistoryServer. LogAggregationService fusionará logs locales generados por containers en un archivo de log y lo subirá al HDFS, reduciendo en cierta medida el número de archivos de log. Sin embargo, en un clúster a gran escala y ocupado, habrá excesivos archivos de log en HDFS después de ejecutarse a largo plazo.

Por ejemplo, si hay 20 nodos, se generan alrededor de 18 millones de archivos de log dentro del período de limpieza predeterminado (15 días), que ocupan alrededor de 18 GB de la memoria de un NameNode y ralentizan la respuesta del sistema de HDFS.

Solo se requiere la lectura y eliminación para los archivos almacenados en HDFS. Por lo tanto, Hadoop Archives se puede utilizar para archivar periódicamente el directorio de los archivos de registro recopilados.

Archivado de logs

El módulo de AggregatedLogArchiveService se agrega al MR HistoryServer para comprobar periódicamente el número de archivos en el directorio log. Cuando el número de archivos alcanza el umbral, AggregatedLogArchiveService inicia una tarea de archivado para archivar archivos de log. Después de archivar, elimina los archivos de log originales para reducir los archivos de log en HDFS.

Limpieza de logs archivados

Hadoop Archives no admite la eliminación en archivos archivados. Por lo tanto, todo el paquete de log de archivo debe eliminarse al limpiar log. El último tiempo de generación de log se obtiene modificando el módulo de AggregatedLogDeletionService. Si todos los archivos de log cumplen los requisitos de limpieza, se puede eliminar el paquete de log de archivo.

Exploración de logs archivados

Hadoop Archives permite el acceso basado en URI al contenido del archivo en el paquete de log de archivo. Por lo tanto, si MR History Server detecta que los archivos de registro originales no existen durante la exploración de archivos, redirige directamente el URI al paquete de log de archivo para acceder al archivo de log archivado.

NOTA

- Esta función invoca Hadoop Archives de HDFS para el archivo de log. Porque la ejecución de una tarea de archivado por Hadoop Archives es ejecutar una aplicación de MR. Por lo tanto, después de ejecutar una tarea de archivado, se agrega un registro de ejecución de MR.
- Esta función de archivar logs se basa en la función de recopilación de log. Por lo tanto, esta función es válida solo cuando la función de recopilación de log está habilitada.

6.23 Oozie

6.23.1 Principios básicos de Oozie

Introducción a Oozie

Oozie es un motor de flujo de trabajo de código abierto que se utiliza para programar y coordinar trabajos de Hadoop.

Arquitectura

El motor de Oozie es una aplicación web integrada en Tomcat por defecto. Oozie usa bases de datos de PostgreSQL.

Oozie proporciona una consola web basada en Ext, a través de la cual los usuarios pueden ver y supervisar los flujos de trabajo de Oozie. Oozie proporciona una API de servicio web REST externa para que el cliente Oozie controle los flujos de trabajo (como iniciar y detener operaciones), y organizar y ejecutar tareas de MapReduce de Hadoop. Para obtener más información, consulte **Figura 6-79**.

Figura 6-79 Arquitectura de Oozie

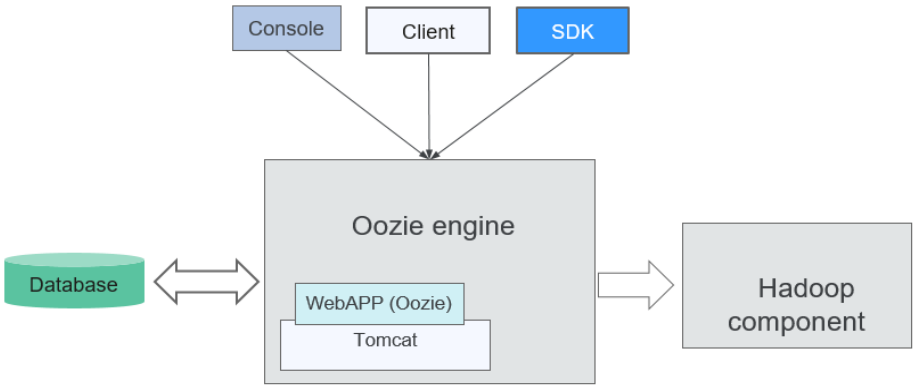


Tabla 6-20 describe las funciones de cada módulo mostrado en **Figura 6-79**.

Tabla 6-20 Descripción de la arquitectura

Nombre de la conexión	Descripción
Console	Permite a los usuarios ver y supervisar los flujos de trabajo de Oozie.

Nombre de la conexión	Descripción
Client	Controla los flujos de trabajo, incluidos los flujos de trabajo de envío, inicio, ejecución, plantación y restauración, a través de API.
SDK	Es la abreviatura de kit de desarrollo de software. Un SDK es un conjunto de herramientas de desarrollo utilizadas por los ingenieros de software para establecer aplicaciones para paquetes de software particulares, marcos de software, plataformas de hardware y sistemas operativos.
Database	Base de datos de PostgreSQL
WebApp (Oozie)	Funciona como el servidor de Oozie. Se puede implementar en un contenedor Tomcat integrado o externo. La información registrada por WebApp (Oozie), incluidos logs se almacena en la base de datos de PostgreSQL.
Tomcat	Un servidor de aplicaciones de web de código abierto gratuito
Hadoop components	Componentes subyacentes, como MapReduce y Hive, que ejecutan los flujos de trabajo orquestados por Oozie.

Principio

Oozie es un servidor de motor de flujo de trabajo que ejecuta flujos de trabajo de MapReduce. También es una aplicación web Java que se ejecuta en un contenedor Tomcat.

Los flujos de trabajo de Oozie se construyen usando Hadoop Process Definition Language (HPDL). HPDL es un lenguaje definido por XML, similar a JBoss jBPM Process Definition Language (jPDL). Un flujo de trabajo de Oozie consiste en el nodo de control y el nodo de acción.

- Control Node controla la orquestación del flujo de trabajo, como **start**, **end**, **error**, **decision**, **fork** y **join**.
- Un flujo de trabajo de Oozie contiene varios nodos de acción, como MapReduce y Java.

  Todos los nodos de acción se implementan y se ejecutan en modo Direct Acyclic Graph (DAG). Por lo tanto, los nodos de acción se ejecutan en dirección. Es decir, el siguiente nodo de acción sólo puede ejecutarse cuando finaliza la ejecución del anterior nodo de acción. Cuando finaliza un nodo de acción, el servidor remoto vuelve a invocar a la interfaz de Oozie. A continuación, Oozie ejecuta el siguiente nodo de acción del flujo de trabajo de la misma manera hasta que se ejecuten todos los nodos de acción (se cuentan los errores de ejecución).

Los flujos de trabajo de Oozie proporcionan varios tipos de nodos de acción, como MapReduce sistemas de archivos distribuidos de Hadoop (HDFS), Secure Shell (SSH), Java y subflujos de Oozie, para soportar una amplia gama de requisitos empresariales.

6.23.2 Funciones de código abierto mejoradas de Oozie

Función de código abierto mejorada: seguridad mejorada

Proporciona roles de administrador y usuarios comunes para admitir la gestión de permisos de Oozie.

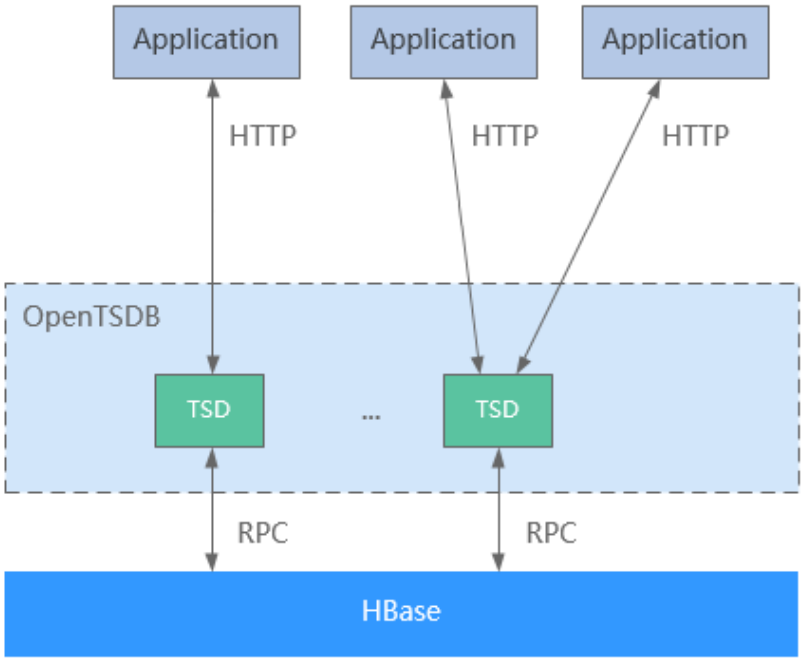
Admite inicio de sesión y salida únicos, acceso a HTTPS y registros de auditoría.

6.24 OpenTSDB

OpenTSDB es una base de datos de series temporales escalable y distribuida basada en HBase. OpenTSDB está diseñado para recopilar información de monitoreo de un clúster a gran escala e implementar consultas de datos de segundo nivel, eliminando las limitaciones de consultas y almacenamiento de cantidades masivas de datos de monitoreo en bases de datos comunes.

OpenTSDB consiste en un Time Series Daemon (TSD) así como un conjunto de utilidades de línea de comandos. La interacción con OpenTSDB se implementa principalmente ejecutando uno o más TSD. Cada TSD es independiente. No hay servidor maestro ni estado compartido, por lo que puede ejecutar tantos TSD como sea necesario para manejar cualquier carga que le arroje. Cada TSD utiliza HBase en un clúster de CloudTable para almacenar y recuperar datos de series temporales. El esquema de datos está altamente optimizado para agregaciones rápidas de series de tiempo similares para minimizar el espacio de almacenamiento. Los usuarios de TSD nunca necesitan acceder directamente al almacenamiento subyacente. Puede comunicarse con el TSD a través de una HTTP API. Todas las comunicaciones ocurren en el mismo puerto (El TSD determina el protocolo del cliente mirando los primeros bytes que recibe).

Figura 6-80 Arquitectura de OpenTSDB



Los escenarios de aplicaciones de OpenTSDB tienen las siguientes características:

- Las métricas recopiladas tienen un valor único en un punto de tiempo y no tienen una estructura o relación compleja.
- Las métricas de monitoreo cambian con el tiempo.
- Al igual que HBase, OpenTSDB ofrece un alto rendimiento y una buena escalabilidad.

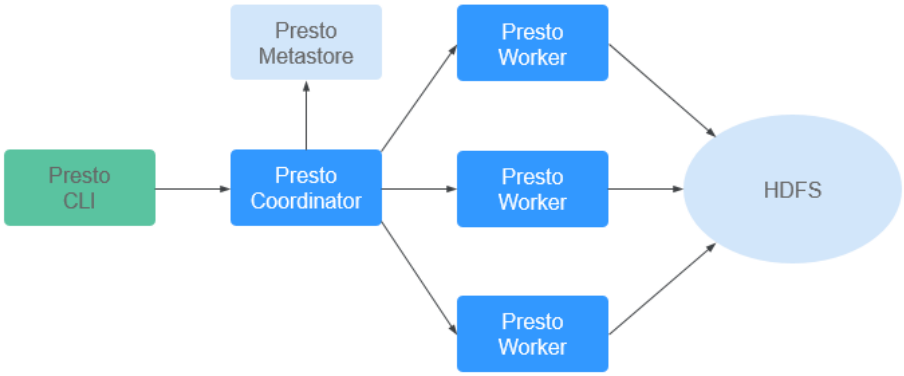
OpenTSDB proporciona una interfaz de programación de aplicaciones basada en HTTP para permitir la integración con sistemas externos. Casi todas las funciones de OpenTSDB son accesibles a través de la API, como la consulta de datos de series de tiempo, la gestión de metadatos y el almacenamiento de puntos de datos. Para más detalles, consulte **https://opentsdb.net/docs/build/html/api_http/index.html**.

6.25 Presto

Presto es un motor de consultas SQL de código abierto para ejecutar consultas analíticas interactivas contra fuentes de datos de todos los tamaños. Se aplica al análisis de datos estructurados/semiestructurados masivos, agregación/informe de datos multidimensionales masivos, ETL, consultas de ad-hoc y más escenarios.

Presto permite consultar datos donde viven, incluidos HDFS, Hive, HBase, Cassandra, bases de datos relacionales o incluso almacenes de datos propietarios. Una consulta de Presto puede combinar diferentes orígenes de datos para realizar análisis de datos en todos los orígenes de datos.

Figura 6-81 Arquitectura de Presto



Presto se ejecuta en un clúster en modo distribuido y contiene un coordinador y varios procesos de trabajo. Las solicitudes de consulta se envían desde los clientes (por ejemplo, CLI) al coordinador. El coordinador analiza las sentencias SQL, genera planes de ejecución y los distribuye a varios procesos de trabajo para su ejecución.

Para obtener más información sobre Presto, visite **https://prestodb.github.io/** o **https://prestosql.io/**.

Múltiples Instancias de Presto

MRS admite la instalación de varias instancias de Presto para un clúster a gran escala de forma predeterminada. Es decir, varias instancias de Worker, tales como Worker1, Worker2 y

Worker3, se instalan en un nodo de Core/Task. Varias instancias de Worker interactúan con el Coordinador para ejecutar tareas informáticas, lo que mejora en gran medida la utilización de recursos de nodo y la eficiencia informática.

Presto de instancias múltiples solo se aplica a la arquitectura de Arm. Actualmente, un único nodo admite un máximo de cuatro instancias.

Para obtener más información sobre la implementación de Presto, consulte **https://prestodb.io/docs/current/installation/deployment.html** o **https://trino.io/docs/current/installation/deployment.html**.

6.26 Ranger

6.26.1 Principios básicos de Ranger

Apache Ranger ofrece un marco de gestión de seguridad centralizado y admite autorización y auditoría unificadas. Gestiona el control de acceso de grano fino sobre Hadoop y componentes relacionados, como Storm, HDFS, Hive, HBase y Kafka. Puede utilizar la consola de interfaz de usuario de web de front-end proporcionada por Ranger para configurar políticas que controlen el acceso de los usuarios a estos componentes.

Figura 6-82 muestra la arquitectura de Ranger.

Figura 6-82 Estructura del Ranger

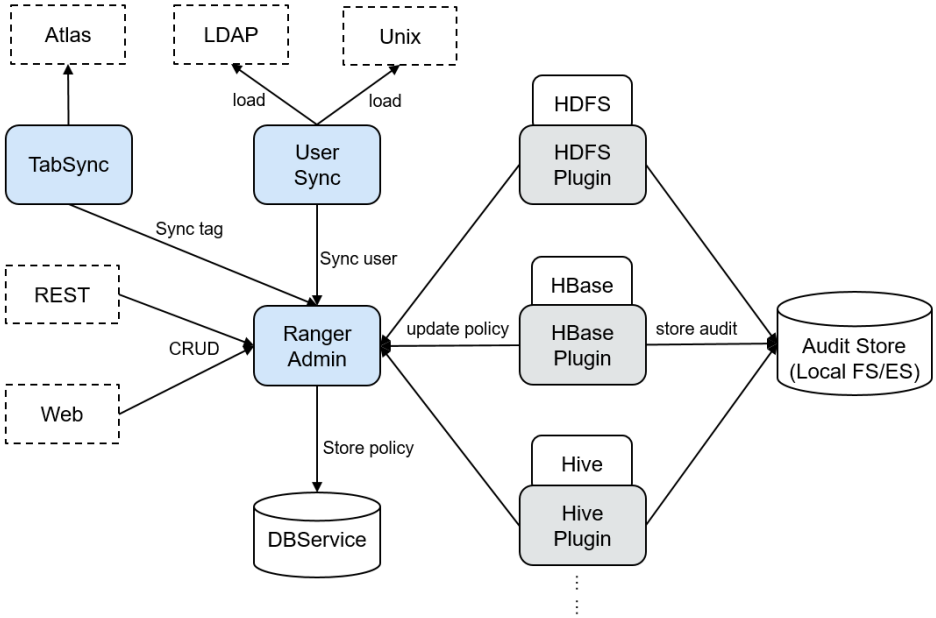


Tabla 6-21 Descripción de la arquitectura

Nombre de la conexión	Descripción
RangerAdmin	Proporciona una API de WebUI y RESTful para gestionar políticas, usuarios y auditorías.

Nombre de la conexión	Descripción
UserSync	Sincroniza periódicamente la información de usuarios y grupos de usuarios de un sistema externo y escribe la información en RangerAdmin.
TagSync	Sincroniza periódicamente la información de las etiquetas del servicio externo de Atlas y escribe la información de las etiquetas en RangerAdmin.

Principios de Ranger

- Complementos de Ranger

  Ranger proporciona complementos de control de acceso basado en políticas (PBAC) para reemplazar los complementos de autenticación originales de los componentes. Los complementos de Ranger se desarrollan en base a la interfaz de autenticación de los componentes. Los usuarios establecen políticas de permisos para los servicios especificados en la interfaz de usuario web de Ranger. Los complementos de Ranger actualizan periódicamente las políticas de la RangerAdmin y las almacenan en caché en el archivo local del componente. Cuando se necesita autenticar una solicitud de cliente, el complemento Ranger hace coincidir el usuario incluido en la solicitud con la política y, a continuación, devuelve un mensaje de aceptación o rechazo.

- Sincronización de usuarios de UserSync

  UserSync sincroniza periódicamente datos de LDAP/Unix a RangerAdmin. En el modo de seguridad, los datos se sincronizan desde LDAP. En modo sin seguridad, los datos se sincronizan desde Unix. De forma predeterminada, se utiliza el modo de sincronización incremental. En cada período de sincronización, UserSync solo actualiza usuarios y grupos de usuarios nuevos o modificados. Cuando se elimina un usuario o un grupo de usuarios, UserSync no sincroniza el cambio con RangerAdmin. Es decir, el usuario o grupo de usuarios no se elimina del RangerAdmin. Para mejorar el rendimiento, UserSync no sincroniza los grupos de usuarios a los que RangerAdmin no pertenece ningún usuario.

- Auditoría unificada

  Los complementos de Ranger pueden grabar registros de auditoría. Actualmente, los registros de auditoría se pueden almacenar en archivos localesBy default, audit logs are stored in local files. To enable Elasticsearch storage, enable it by following the instructions provided in the guide and query the audit details of the corresponding components on the Audit tab page of Ranger WebUI.

- Alta confiabilidad

  Ranger admite dos RangerAdmins que funcionan en modo activo/activo. Dos RangerAdmins prestan servicios al mismo tiempo. Si cualquiera de los RangerAdmin es defectuoso, Ranger sigue trabajando.

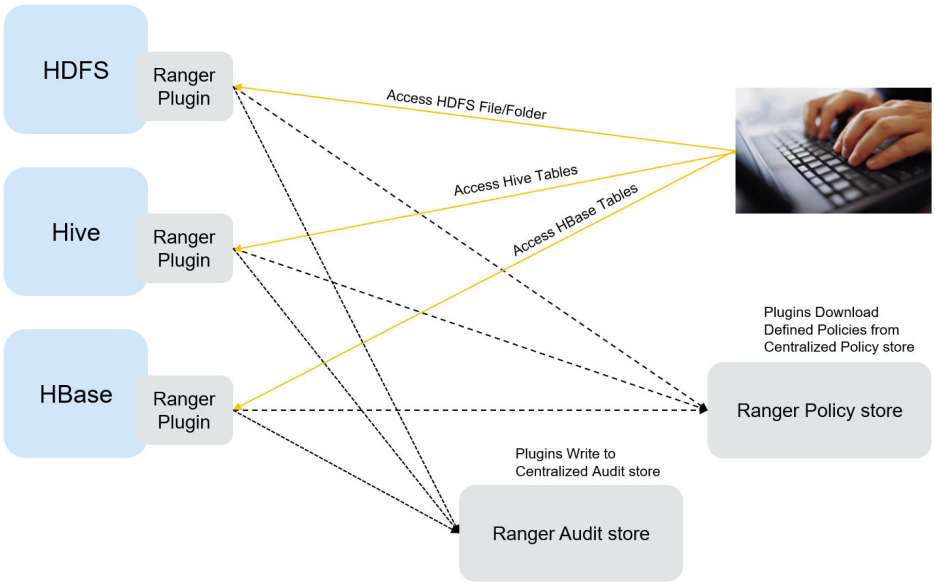
- Alto rendimiento

  Ranger proporciona la capacidad de balanceo de carga. Cuando un usuario accede a Ranger WebUI usando un navegador, el balanceo de carga selecciona automáticamente el RangerAdmin con la carga más ligera para proporcionar servicios.

6.26.2 Relación entre Ranger y otros componentes

Ranger proporciona complementos de autenticación basados en PABC para que los componentes se ejecuten en sus servidores. Ranger actualmente admite la autenticación para los siguientes componentes como HDFS, YARN, Hive, HBase, Kafka, Storm, y Spark2x. Más componentes serán compatibles en el futuro.

Figura 6-83 Relación entre Ranger y otros componentes



6.27 Spark

6.27.1 Principios básicos de Spark

NOTA

El componente de Spark se aplica a versiones anteriores a MRS 3.x.

Descripción

Spark es un marco de procesamiento de datos paralelo de código abierto. Le ayuda a desarrollar fácilmente aplicaciones de big data unificadas y a realizar procesamiento fuera de línea, procesamiento de flujo y análisis interactivo de datos.

Spark proporciona un marco de trabajo que incluye computación rápida, escritura y consultas interactivas. Spark tiene ventajas obvias sobre Hadoop en términos de rendimiento. Spark utiliza el modo de computación en memoria para evitar cuellos de botella de E/S en escenarios donde varias tareas en un flujo de trabajo de MapReduce procesan el mismo conjunto de datos. Spark se implementa mediante el uso del lenguaje de programación Scala. Scala permite procesar conjuntos de datos distribuidos en un método que es el mismo que el de procesar datos locales. Además del análisis de datos interactivo, Spark admite la minería de datos interactiva. Spark adopta la informática en memoria, lo que facilita la informática iterativa. Por coincidencia, la computación iterativa de los mismos datos es un problema general que enfrenta la minería de datos. Además, Spark puede ejecutarse en clústeres de Yarn

donde está instalado Hadoop 2.0. La razón por la que Spark no solo puede conservar varias características como MapReduce tolerancia a fallos, localización de datos y escalabilidad, sino también garantizar un alto rendimiento y evitar E/S de disco ocupado es que se crea una estructura de abstracción de memoria llamada Conjunto de datos distribuidos resilientes (RDD) para Spark.

La abstracción de memoria distribuida original, por ejemplo, el almacén de valores clave y las bases de datos, admite una actualización de pequeña granularidad del estado de las variables. Esto requiere una copia de respaldo de datos o actualizaciones de registro para garantizar la tolerancia a fallas. En consecuencia, se produce una gran cantidad de consumo de E/S en flujos de trabajo intensivos en datos. Para el RDD, solo tiene un conjunto de API restringidas y solo admite actualizaciones de gran granularidad, por ejemplo, map y join. De esta manera, Spark solo necesita registrar los registros de operaciones de transformación generados durante el establecimiento de datos para garantizar la tolerancia a fallas sin registrar un conjunto de datos completo. Este registro de enlace de transformación de datos es una fuente para rastrear un conjunto de datos. En general, las aplicaciones paralelas aplican el mismo proceso informático para un conjunto de datos grande. Por lo tanto, el límite a la mencionada actualización de granularidad no es grande. Como se describe en las tesis de Spark, el RDD puede funcionar como múltiples marcos de computación diferentes, por ejemplo, modelos de programación de MapReduce y Pregel. Además, Spark le permite hacer que un proceso de transformación de datos sea explícitamente persistente en los discos duros. La localización de datos se implementa al permitirle controlar particiones de datos en función del valor clave de cada registro. (Una ventaja obvia de este método es que dos copias de datos a asociar serán hashed en el mismo modo.) Si el uso de memoria excede el límite físico, Spark escribe particiones relativamente grandes en los discos duros, lo que garantiza la escalabilidad.

Spark tiene las siguientes características:

- Rápido: La velocidad de procesamiento de datos de Spark es de 10 a 100 veces mayor que la de MapReduce.
- Fácil de usar: Java, Scala y Python se pueden usar para compilar aplicaciones paralelas de manera sencilla y rápida para procesar cantidades masivas de datos. Spark ofrece más de 80 operadores para ayudarle a compilar aplicaciones paralelas.
- Universal: Spark proporciona muchas herramientas, por ejemplo, **Spark SQL** y **Spark Streaming**. Estas herramientas se pueden combinar de forma flexible en una aplicación.
- Integración con Hadoop: Spark puede ejecutarse directamente en un clúster de Hadoop y leer datos de Hadoop existentes.

El componente Spark de MRS tiene las siguientes ventajas:

- El componente de Spark Streaming de MRS admite el procesamiento de datos en tiempo real en lugar de activarse según lo programado.
- El componente de Spark de MRS proporciona Streaming Estructurado y le permite crear aplicaciones de streaming mediante la API de Dataset. Spark admite la semántica de exactly-once y joins internas y externas para los flujos.
- El componente Spark de MRS utiliza **pandas_udf** para reemplazar las funciones definidas por el usuario (UDF) originales de PySpark para procesar datos, lo que reduce la duración de procesamiento de un 60% a un 90% (afectada por operaciones específicas).
- El componente de Spark de MRS también admite el procesamiento de datos de gráficos y permite el modelado utilizando gráficos durante el cálculo de gráficos.

- Spark SQL de MRS es compatible con alguna sintaxis de Hive (basado en las 64 sentencias SQL del conjunto de pruebas Hive-Test-benchmark) y sintaxis SQL estándar (basado en las 99 sentencias SQL del conjunto de pruebas TPC-DS).

Para obtener más información sobre la arquitectura y los principios de Spark, visite **https://spark.apache.org/docs/3.1.1/quick-start.html**.

Arquitectura

Figura 6-84 describe la arquitectura de Spark y **Tabla 6-22** enumera los módulos de Spark.

Figura 6-84 Arquitectura de Spark

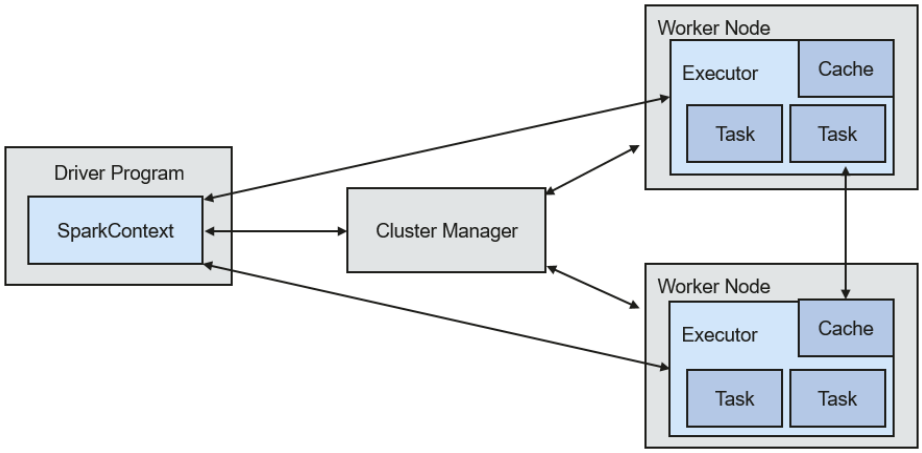


Tabla 6-22 Conceptos básicos

Módulo	Descripción
Cluster Manager	El administrador de clústeres gestiona los recursos del clúster. Spark admite varios administradores de clústeres, incluidos Mesos, Yarn y el administrador de clústeres independiente que se entrega con Spark.
Application	Aplicación de Spark. Consiste en un programa de Driver Program y múltiples executors.
Deploy Mode	Implementación en modo de clúster o cliente. En modo de clúster, el controlador se ejecuta en un nodo dentro del clúster. En modo cliente, el controlador se ejecuta en el cliente (fuera del clúster).
Driver Program	El proceso principal de la aplicación Spark. Ejecuta la función **main()** de una aplicación y crea "SparkContext". Se utiliza para analizar aplicaciones, generar etapas y programar tareas a ejecutores. Por lo general, SparkContext representa Driver Program.
Executor	Un proceso iniciado en un Work Node. Se utiliza para ejecutar tareas, y gestionar y procesar los datos utilizados en las aplicaciones. Una aplicación de Spark generalmente contiene varios executors. Cada executor recibe comandos del driver y ejecuta una o varias tareas.

Módulo	Descripción
Worker Node	Nodo que inicia y gestiona ejecutores y recursos en un clúster.
Job	Un job consta de varias tareas simultáneas. Un operador de action (por ejemplo, un operador de collect) se asigna a un job.
Stage	Cada job consta de múltiples etapas. Cada etapa es un conjunto de tareas, que está separado por el Gráfico Acíclico Dirigido (DAG).
Task	Una tarea lleva la unidad de cálculo de la lógica de servicio. Es la unidad de trabajo mínima que se puede ejecutar en la plataforma de Spark. Una aplicación se puede dividir en múltiples tareas basadas en el plan de ejecución y la cantidad de cálculo.

Principio de ejecución de la aplicación de Spark

Figura 6-85 muestra la arquitectura en ejecución de la aplicación de Spark. El proceso en ejecución es el siguiente:

1. Una aplicación se está ejecutando en el clúster como una colección de procesos. El controlador coordina la ejecución de la aplicación.
2. Para ejecutar una aplicación, el controlador se conecta al administrador de clústeres (como Standalone, Mesos y Yarn) para solicitar los recursos del ejecutor e iniciar ExecutorBackend. El administrador de clúster programa los recursos entre diferentes aplicaciones. El controlador programa los DAGs, divide las etapas y genera tareas para la aplicación al mismo tiempo.
3. A continuación, Spark envía los códigos de la aplicación (los códigos transferidos a **SparkContext** definidos por JAR o Python) a executor.
4. Una vez terminadas todas las tareas, se detiene la ejecución de la aplicación de usuario.

Figura 6-85 Arquitectura de ejecución de aplicaciones de Spark

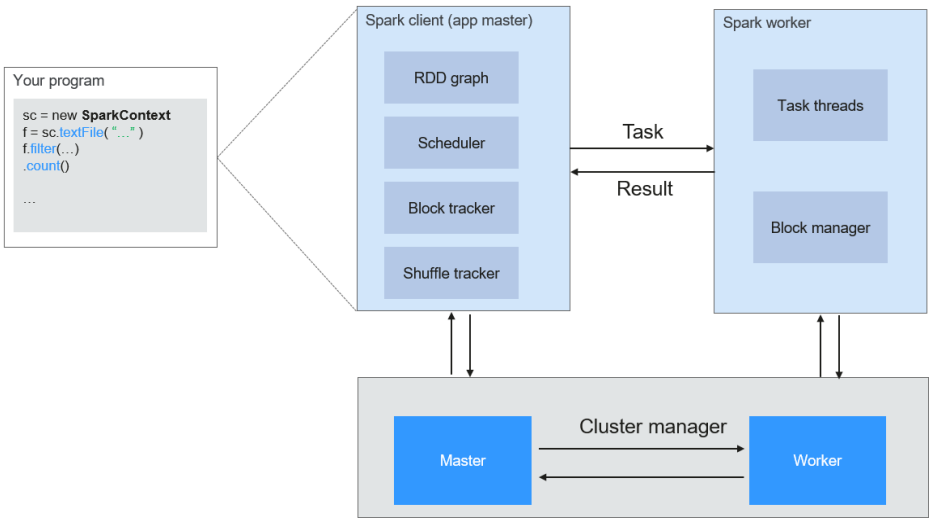
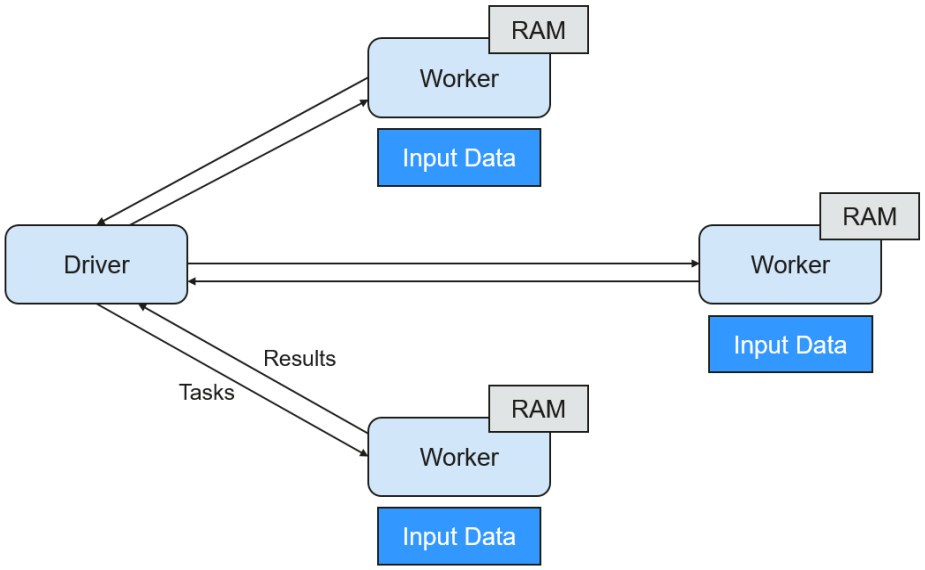


Figura 6-86 muestra los modos de Master y Worker adoptados por Spark. Un usuario envía una aplicación en el cliente de Spark y, a continuación, el programador divide un job en varias

tareas y envía las tareas a cada Worker para su ejecución. Cada Worker informa los resultados del cálculo al Driver (Master) y, a continuación, el Driver agrega y devuelve los resultados al cliente.

Figura 6-86 Modo de Spark Master-Worker



Tenga en cuenta lo siguiente acerca de la arquitectura:

- Las aplicaciones se aíslan entre sí.

  Cada aplicación tiene un proceso de executor independiente, y cada executor inicia varios subprocesos para ejecutar tareas en paralelo. Ya sea en términos de programación o la ejecución de tarea en executors. Cada driver programa de forma independiente sus propias tareas. Diferentes tareas de aplicación se ejecutan en diferentes JVMs, es decir, diferentes ejecutors.
- Las diferentes aplicaciones de Spark no comparten datos, a menos que los datos se almacenen en el sistema de almacenamiento externo, como HDFS.
- Se recomienda desplegar el programa de Driver en una ubicación cercana al nodo de Worker porque el programa de Driver programa tareas en el clúster. Por ejemplo, desplegar el programa de Driver en la red donde se encuentra el nodo de Worker.

Spark en YARN se puede desplegar en dos modos:

- En el modo de Yarn-cluster, el driver de Spark se ejecuta dentro de un proceso de ApplicationMaster que es getionado por Yarn en el clúster. Después de iniciar el ApplicationMaster, el cliente puede salir sin interrumpir la ejecución del servicio.
- En el modo Yarn-client, el driver se inicia en el proceso de cliente y el proceso de ApplicationMaster solo se utiliza para solicitar recursos del clúster de Yarn.

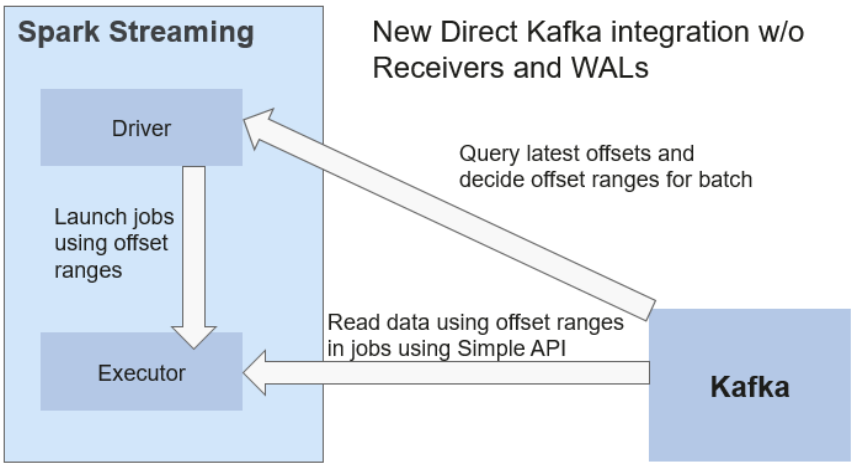
Principio de Spark Streaming

Spark Streaming es un marco informático en tiempo real basado en Spark, que amplía la capacidad para procesar datos de streaming masivos. Actualmente, Spark admite los siguientes métodos de procesamiento de datos:

- Direct Streaming

En el enfoque de Direct Streaming, Direct API se utiliza para procesar datos. Tomemos Kafka Direct API como ejemplo. Direct API proporciona una ubicación de desplazamiento desde la que cada rango de batch leerá, lo que es mucho más simple que iniciar un receptor para recibir continuamente datos de Kafka y datos escritos en registros de escritura previa (WAL). A continuación, cada job de batch se está ejecutando y los datos de desplazamiento correspondientes están listos en Kafka. Esta información de desplazamiento puede almacenarse de forma segura en el archivo de punto de control y leerse por aplicaciones que no se iniciaron.

Figura 6-87 Transmisión de datos a través de Direct Kafka API



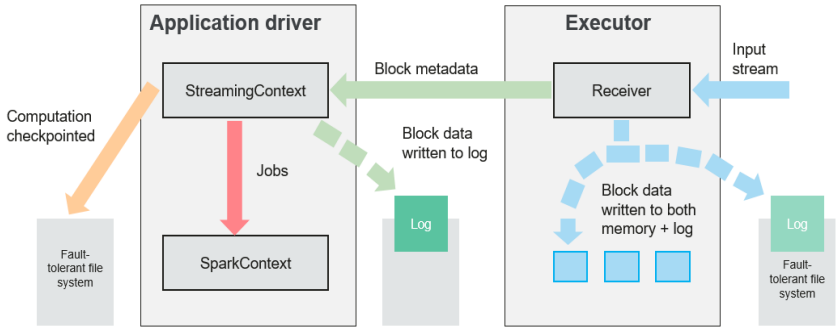
Después de la falla, Spark Streaming puede leer datos de Kafka de nuevo y procesar el segmento de datos. El resultado del procesamiento es el mismo sin importar que Spark Streaming falle o no, porque la semántica se procesa solo una vez.

Direct API no necesita usar el WAL y Receivers, y se asegura de que cada registro de Kafka se reciba solo una vez, lo que es más eficiente. De esta manera, Spark Streaming y Kafka se pueden integrar bien, haciendo que los canales de streaming se presenten con alta tolerancia a fallas, alta eficiencia y facilidad de uso. Por lo tanto, se recomienda utilizar Direct Streaming para procesar datos.

- Receiver

  Cuando se inicia una aplicación de Spark Streaming (es decir, cuando se inicia el driver), el StreamingContext relacionado (la base de todas las funciones de streaming) utiliza SparkContext para iniciar el receiver y convertirse en una tarea en ejecución a largo plazo. Estos receivers reciben y guardan datos de transmisión en la memoria de Spark para su procesamiento. **Figura 6-88** muestra el ciclo de vida de la transferencia de datos.

Figura 6-88 Ciclo de vida de la transferencia de datos



a. Recibir datos (flecha azul).

   Receiver divide un flujo de datos en una serie de bloques y los almacena en la memoria del executor. Además, después de habilitar WAL, escribe datos en el WAL del sistema de archivos tolerante a fallas.

b. Notificar al driver (flecha verde).

   Los metadatos en el bloque recibido se envían a StreamingContext en el driver. Los metadatos incluyen:

   - ID de referencia de bloque utilizado para localizar la posición de los datos en la memoria del Executor.
   - Información de desplazamiento de datos de bloque en registros (si la función WAL está habilitada).

c. Procesar datos (flecha roja).

   Para cada lote de datos, StreamingContext utiliza información de bloque para generar conjuntos de datos distribuidos resilientes (RDD) y jobs. StreamingContext ejecuta jobs ejecutando tareas para procesar bloques en la memoria del executor.

d. Establecer periódicamente puntos de control (flechas naranjas).

   Para la tolerancia a fallas, StreamingContext establece periódicamente puntos de comprobación y los guarda en sistemas de archivos externos.

Tolerancia a fallas

Spark y su RDD permiten procesar sin problemas las fallas de cualquier nodo de Worker en el clúster. Spark Streaming está construido sobre Spark. Por lo tanto, el nodo Worker de Spark Streaming también tiene la misma capacidad de tolerancia a fallas. Sin embargo, Spark Streaming necesita funcionar correctamente en caso de que se ejecute durante mucho tiempo. Por lo tanto, Spark debe ser capaz de recuperarse de fallas a través del proceso del driver (proceso principal que coordina a todos los Workers). Esto plantea desafíos a la tolerancia a fallas del Spark driver debido a que el Spark driver puede ser cualquier aplicación de usuario implementada en cualquier modo de computación. Sin embargo, Spark Streaming tiene una arquitectura de computación interna. Es decir, ejecuta periódicamente el mismo computación de Spark en cada lote de datos. Dicha arquitectura le permite almacenar periódicamente puntos de control en un espacio de almacenamiento confiable y recuperarlos al reiniciar el Driver.

Para los datos de origen tales como archivos, el mecanismo de recuperación del Driver puede garantizar cero pérdida de datos porque todos los datos se almacenan en un sistema de archivos tolerante a fallas como HDFS. Sin embargo, para otras fuentes de datos tales como Kafka y Flume, algunos datos recibidos se almacenan en caché solo en la memoria y pueden perderse antes de ser procesados. Esto es causado por el modo de operación de distribución de las aplicaciones de Spark. Cuando el proceso del driver falla, todos los executors que se ejecutan en el Cluster Manager, junto con todos los datos de la memoria, se terminan. Para evitar dicha pérdida de datos, la función WAL se agrega a Spark Streaming.

WAL se utiliza a menudo en bases de datos y sistemas de archivos para garantizar la persistencia de cualquier operación de datos. Es decir, primero registrar una operación en un registro persistente y realizar esta operación en los datos. Si la operación falla, el sistema se recupera leyendo el registro y volviendo a aplicar la operación preestablecida. A continuación se describe cómo utilizar WAL para garantizar la persistencia de los datos recibidos:

Receiver se utiliza para recibir datos de fuentes de datos tales como Kafka. Como una tarea que se ejecuta durante mucho tiempo en Executor, Receiver recibe datos y también confirma los datos recibidos si son compatibles con los orígenes de datos. Los datos recibidos se

almacenan en la memoria de Executor, y Driver entrega una tarea al Executor para su procesamiento.

Después de habilitar WAL, todos los datos recibidos se almacenan en archivos de registro en el sistema de archivos tolerante a fallas. Por lo tanto, los datos recibidos no pierden incluso si Spark Streaming falla. Además, el receiver comprueba la exactitud de los datos recibidos solo después de que los datos se hayan escrito previamente en registros. Los datos almacenados en caché pero no almacenados pueden ser enviados de nuevo por los orígenes de datos después de que el driver se reinicie. Estos dos mecanismos aseguran cero pérdida de datos. Es decir, todos los datos se recuperan de los registros o se reenvían por las fuentes de datos.

Para habilitar la función WAL, realice las siguientes operaciones:

- Establezca **streamingContext.checkpoint** para configurar el directorio de checkpoint, que es una ruta de archivo HDFS utilizada para almacenar checkpoints de streaming y WAL.
- Establezca **spark.streaming.receiver.writeAheadLog.enable** de SparkConf a **true** (el valor predeterminado es de **false**).

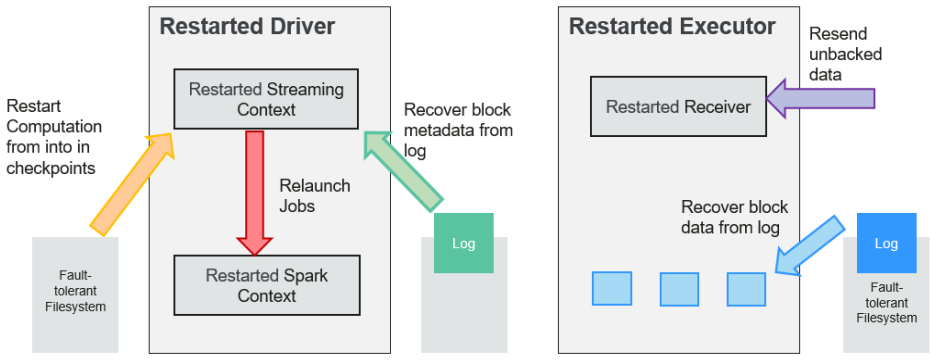
Después de que WAL está habilitado, todos los receivers tienen la ventaja de recuperarse de datos recibidos confiables. Se recomienda desactivar el mecanismo de réplica múltiple porque el sistema de archivos tolerante a errores de WAL también puede replicar los datos.

> **NOTA**
>
> El rendimiento de recepción de datos se reduce después de que se habilita WAL. Todos los datos se escriben en el sistema de archivos tolerante a fallas. Como resultado, el rendimiento de escritura del sistema de archivos y el ancho de banda de red para la replicación de datos puede convertirse en el cuello de botella potencial. Para resolver este problema, se recomienda crear más receptores para aumentar el grado de paralelismo de recepción de datos o utilizar mejor hardware para mejorar el rendimiento del sistema de archivos tolerante a fallas.

Proceso de recuperación

Cuando se reinicia un driver fallido, reinícielo de la siguiente manera:

Figura 6-89 Proceso de recuperación de computación



1. Recuperar la computación. (flecha naranja)

   Utilice la información del punto de control para reiniciar Driver, reconstruir el SparkContext y reinicia Receiver.

2. Recuperar el bloque de metadatos. (flecha verde)

   Esta operación asegura que todos los bloques de metadatos necesarios se recuperan para continuar con la recuperación informática posterior.

3. Relanzar trabajos inacabados. (flecha roja)

   Los metadatos recuperados se utilizan para generar RDD y trabajos correspondientes para el procesamiento por lotes interrumpido debido a fallas.

4. Leer los datos del bloque guardados en los registros. (flecha azul)

   Los datos de bloque se leen directamente de los WAL durante la ejecución de los trabajos anteriores y, por lo tanto, se recuperan todos los datos esenciales almacenados de forma confiable en los registros.

5. Reenviar datos no confirmados. (flecha púrpura)

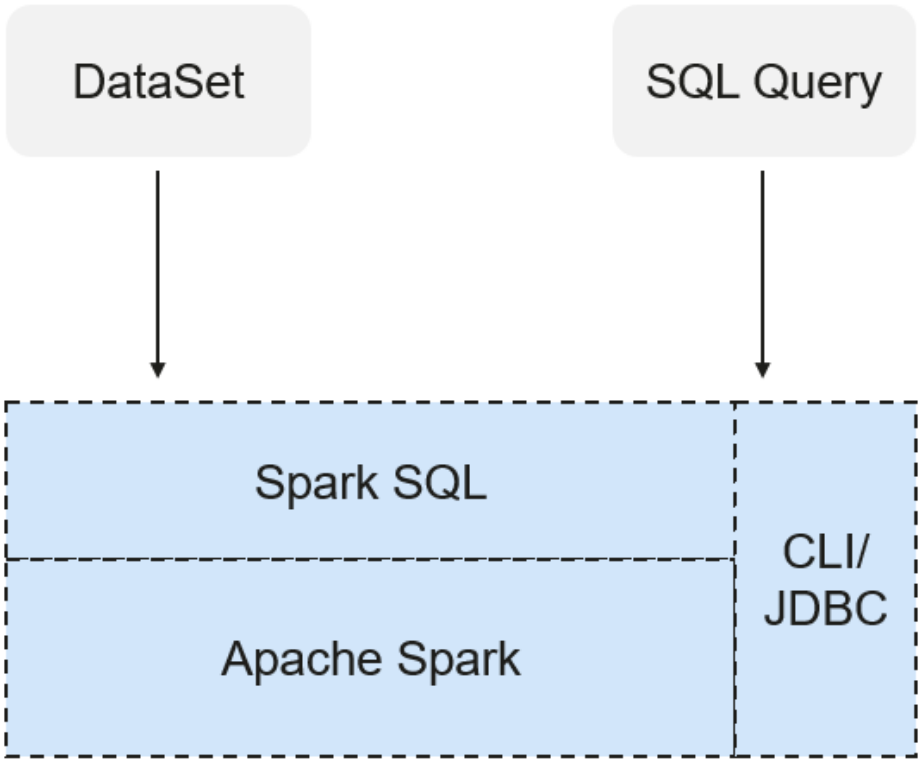
   Los datos que se almacenan en caché pero no se almacenan en los registros en caso de fallas son reenviados por las fuentes de datos, porque el receiver no confirma los datos.

Por lo tanto, mediante el uso de WALs y Receiver confiable, Spark Streaming puede evitar la pérdida de datos de entrada causada por fallas de Driver.

Principio de SparkSQL y DataSet

SparkSQL

Figura 6-90 SparkSQL y DataSet



Spark SQL es un módulo para procesar datos estructurados. En la aplicación de Spark, las sentencias de SQL o las API de DataSet se pueden usar sin problemas para consultar datos estructurados.

Spark SQL y DataSet también proporcionan un método universal para acceder a múltiples fuentes de datos como Hive, CSV, Parquet, ORC, JSON y JDBC. Estas fuentes de datos también permiten la interacción de datos. Spark SQL reutiliza la lógica de procesamiento de

frontend Hive y el módulo de procesamiento de metadatos. Con Spark SQL, puede consultar directamente los datos de Hive existentes.

Además, Spark SQL también proporciona API, CLI y API JDBC, lo que permite diversos accesos al cliente.

Spark SQL Native DDL/DML

En Spark 1.5, muchos comandos de Lenguaje de definición de datos (DDL)/Lenguaje de manipulación de datos (DML) se presionan hacia abajo y se ejecutan en Hive, causando acoplamiento con Hive e inflexibilidad, como informes de errores inesperados y resultados.

Spark 3.1.1 realiza la localización de comandos y reemplaza Hive con Spark SQL Native DDL/DML para ejecutar comandos de DDL/DML. Además, se realiza el desacoplamiento de Hive y se pueden personalizar los comandos.

DataSet

Un DataSet es una colección fuertemente tipada de objetos específicos de dominio que se pueden transformar en paralelo usando operaciones funcionales o relacionales. Cada Dataset también tiene una vista no escrita llamada DataFrame, que es un Dataset de Row.

El DataFrame es un conjunto de datos estructurado y distribuido que consta de varias columnas. El DataFrame es igual a una tabla en la base de datos de relaciones o al DataFrame en R/Python. El DataFrame es el concepto más básico en Spark SQL, que se puede crear mediante múltiples métodos, como el conjunto de datos estructurado, la tabla Hive, la base de datos externa o RDD.

Las operaciones disponibles en el DataSets se dividen en transformaciones y acciones.

- Una operación de transformation puede generar un nuevo DataSet,

  por ejemplo **map**, **filter**, **select** y **aggregate (groupBy)**.
- Una operación de action puede desencadenar el cálculo y devolver resultados,

  por ejemplo **count**, **show** o escribir datos en el sistema de archivos.

Puede utilizar cualquiera de los siguientes métodos para crear un DataSet:

- La forma más común es apuntar Spark a algunos archivos en sistemas de almacenamiento, usando la función **read** disponible en un SparkSession.
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people =
  spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- También puede crear un DataSet mediante la operación de transformation disponible en una existente.

  Por ejemplo, aplique la operación de map en un DataSet existente para crear un DataSet:
  ```
  val names = people.map(_.name) // In Scala: names is Dataset.
  Dataset<String> names = people.map((Person p) -> p.name,
  Encoders.STRING)); // Java
  ```

CLI y JDBCServer

Además de las API de programación, Spark SQL también proporciona las API de CLI/JDBC.

- Los scripts **spark-shell** y **spark-sql** pueden proporcionar la CLI para la depuración.
- JDBCServer proporciona API de JDBC. Los sistemas externos pueden enviar directamente solicitudes JDBC para calcular y analizar datos estructurados.

Principio de SparkSession

SparkSession es una API unificada para la programación de Spark y puede considerarse como una entrada unificada para la lectura de datos. SparkSession proporciona un único punto de entrada para realizar muchas operaciones que anteriormente estaban dispersas entre varias clases, y también proporciona métodos de acceso a estas clases más antiguas para maximizar la compatibilidad.

Un SparkSession se puede crear usando un patrón de constructor. El constructor reutilizará automáticamente el SparkSession existente si hay un SparkSession o creará un SparkSession si no existe. Durante las transacciones de E/S, la configuración del elemento de configuración en el generador se sincroniza automáticamente con Spark y Hadoop.

```
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder
  .master("local")
  .appName("my-spark-app")
  .config("spark.some.config.option", "config-value")
  .getOrCreate()
```

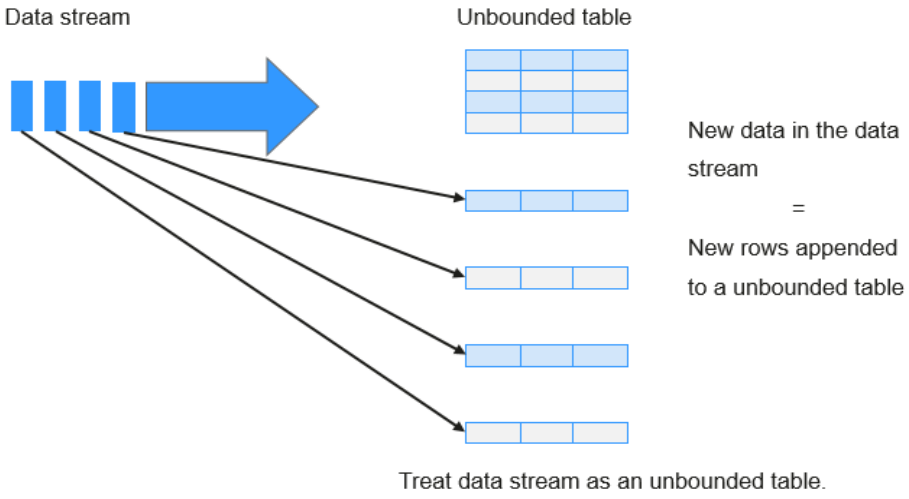
- SparkSession se puede usar para ejecutar consultas de SQL sobre datos y devolver resultados como DataFrame.
  ```
  sparkSession.sql("select * from person").show
  ```
- SparkSession se puede utilizar para establecer elementos de configuración durante la ejecución. Estos elementos de configuración se pueden reemplazar con variables en las sentencias de SQL.
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- SparkSession también incluye un método de "catalog" que contiene métodos para trabajar con Metastore (catálogo de datos). Después de utilizar este método, se devuelve un conjunto de datos, que se puede ejecutar utilizando la misma API de conjunto de datos.
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```
- Se puede acceder a SparkContext subyacente mediante la API de SparkContext de SparkSession.
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Principio de Structured Streaming

El Structured Streaming es un motor de procesamiento de flujo construido en el motor de Spark SQL. Puede usar la API Dataset/DataFrame en Scala, Java, Python o R para expresar agregaciones de streaming, ventanas de tiempo de eventos y stream-stream joins. Si los datos de streaming se producen de forma incremental y continua, Spark SQL continuará procesando los datos y sincronizando el resultado con el conjunto de resultados. Además, el sistema garantiza una tolerancia a fallas de extremo a extremo exactamente una vez a través de checkpoints y WALs.

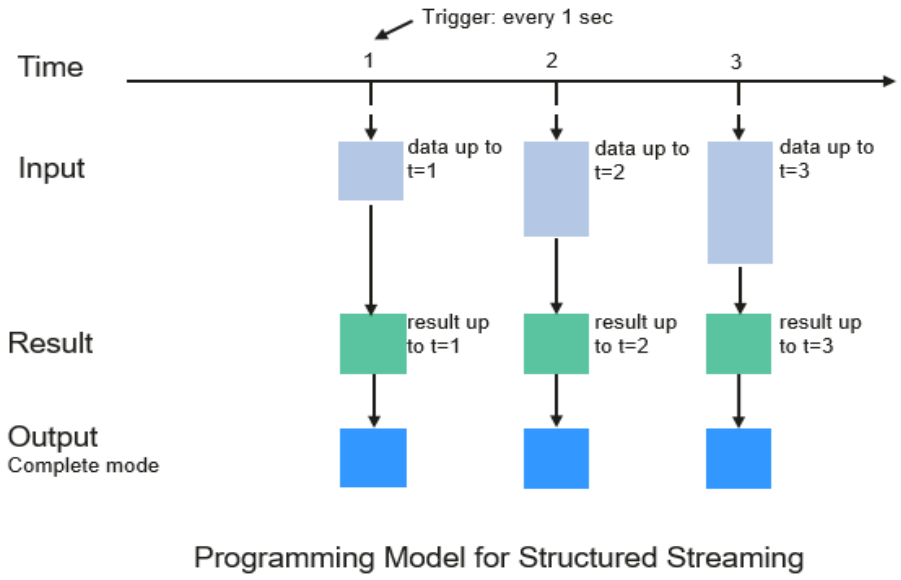
El núcleo del Structured Streaming es tomar los datos de streaming como una tabla de base de datos incremental. De manera similar al modelo de procesamiento de bloques de datos, el modelo de procesamiento de datos de flujo continuo aplica operaciones de consulta en una tabla de base de datos estática a la computación de flujo continuo, y Spark usa sentencias SQL estándar para consulta, para obtener datos de la tabla incremental y no delimitada.

Figura 6-91 Tabla no delimitada de Structured Streaming



Cada operación de consulta generará una tabla de resultados. En cada intervalo de activación, los datos actualizados se sincronizarán con la tabla de resultados. Cada vez que se actualiza la tabla de resultados, el resultado actualizado se escribirá en un sistema de almacenamiento externo.

Figura 6-92 Modelo de procesamiento de datos de Structured Streaming



Los modos de almacenamiento de Structured Streaming en la fase de output son los siguientes:

- Complete Mode: los conjuntos de resultados actualizados se escriben en el sistema de almacenamiento externo. La operación de escritura se realiza por un conector del sistema de almacenamiento externo.

- Append Mode: Si se activa un intervalo, solo los datos agregados en la tabla de resultados se escribirán en un sistema externo. Esto solo se aplica a las consultas en las que no se espera que cambien las filas existentes en la tabla de resultados.
- Update Mode: si se activa un intervalo, solo los datos actualizados en la tabla de resultados se escribirán en un sistema externo, que es la diferencia entre el Complete Mode y Update Mode.

Conceptos básicos

- **RDD**

  El conjunto de datos distribuidos resilientes (RDD) es un concepto central de Spark. Indica un conjunto de datos distribuido de solo lectura y particionado. Los datos parciales o todos de este conjunto de datos pueden almacenarse en caché en la memoria y reutilizarse entre computaciones.

  Creación de RDD

  - Se puede crear un RDD a partir de la entrada de HDFS u otros sistemas de almacenamiento que sean compatibles con Hadoop.
  - Un nuevo RDD se puede convertir a partir de un RDD principal.
  - Un RDD se puede convertir a partir de una colección de conjuntos de datos a través de la codificación.

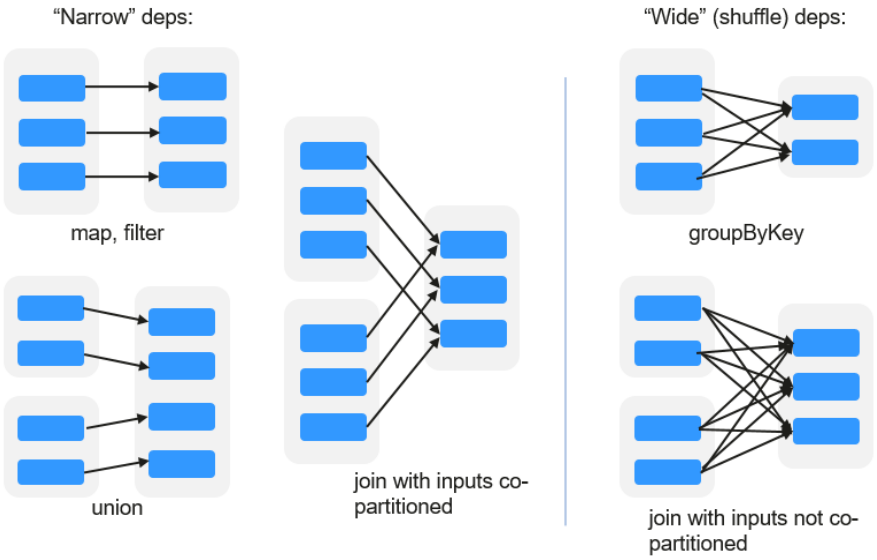
  Almacenamiento de RDD

  - Puede seleccionar diferentes niveles de almacenamiento para almacenar un RDD para su reutilización. (Hay 11 niveles de almacenamiento para almacenar un RDD.)
  - Por defecto, el RDD se almacena en la memoria. Cuando la memoria es insuficiente, el RDD se desborda al disco.

- **Dependencia de RDD**

  La dependencia de RDD incluye la dependencia estrecha y la dependencia amplia.

  Figura 6-93 Dependencia de RDD

  - **Dependencia estrecha**: Cada partición del RDD principal es utilizada como máximo por una partición del RDD secundario.

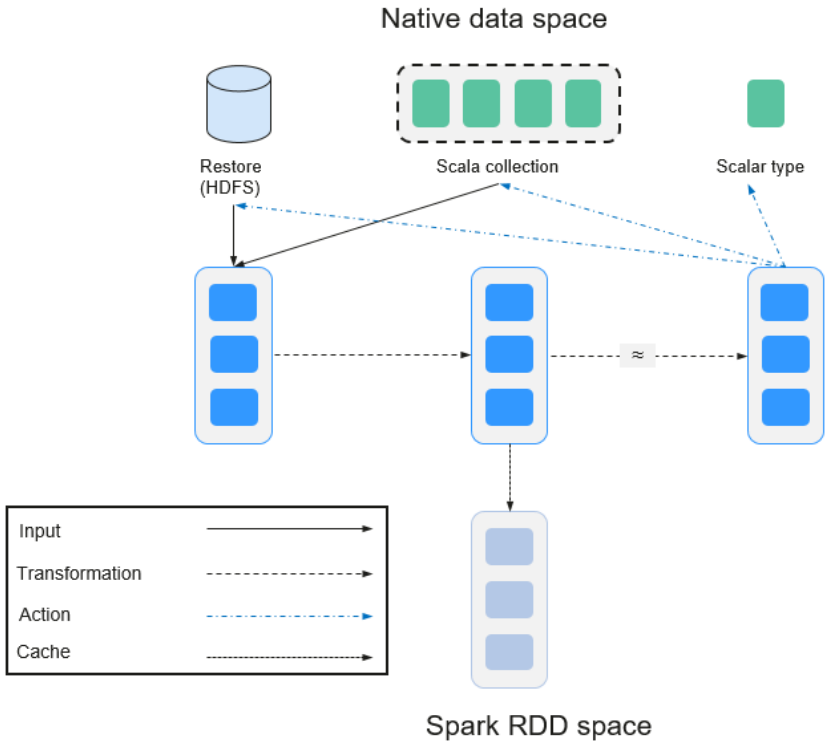
- **Dependencia amplia**: Las particiones del RDD secundaria dependen de todas las particiones del RDD principal.

La estrecha dependencia facilita la optimización. Lógicamente, cada operador de RDD es un fork/join (el join no es el operador de join mencionado anteriormente, sino la barrier utilizada para sincronizar múltiples tareas simultáneas); fork el RDD a cada partición y, a continuación, realiza el cálculo. Después de la computación, join los resultados y, a continuación, realice la operación de fork/join en el siguiente operador de RDD. Es antieconómico traducir directamente el RDD en implementación física. El primero es que cada RDD (incluso resultado intermedio) necesita ser físico en memoria o almacenamiento, lo que consume mucho tiempo y ocupa mucho espacio. El segundo es que, como barrier global, la operación de join es muy costosa y todo el proceso de join se ralentizará por el nodo más lento. Si las particiones del RDD secundaria dependen estrechamente de la del RDD principal, los dos procesos fork/join se pueden combinar para implementar la optimización de fusión clásica. Si la relación en la secuencia continua del operador es una dependencia estrecha, se pueden combinar múltiples procesos de fork/join para reducir un gran número de barriers globales y eliminar la fisicalización de muchos resultados intermedios de RDD, lo que mejora enormemente el rendimiento. Esto se llama optimización de canalización en Spark.

- **Transformation y Action (Operaciones de RDD)**

  Las operaciones en RDD incluyen transformation (el valor devuelto es un RDD) y action (el valor devuelto no es un RDD). **Figura 6-94** muestra el proceso de operación de RDD. El transformation es lazy, lo que indica que la transformación de un RDD a otro RDD no se ejecuta inmediatamente. Spark solo registra la transformation, pero no la ejecuta inmediatamente. La computación real se inicia solo cuando se inicia la acción. El action devuelve los resultados o escribe los datos RDD en el sistema de almacenamiento. Action es la fuerza motriz para que Spark inicie la computación.

Figura 6-94 Operación de RDD



Los datos y el modelo de operación de RDD son bastante diferentes de los de Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

a. El operador textFile lee los archivos de registro del HDFS y devuelve files (como un RDD).

b. El operador de filtro filtra las filas con **ERROR** y las asigna a errors (un nuevo RDD). El operador de filter es una transformation.

c. El operador de cache almacena en caché los errores para su uso futuro.

d. El operador de count devuelve el número de filas de errores. El operador de count es una acción.

Transformation incluye los siguientes tipos:

- Los elementos RDD se consideran elementos simples.

  La entrada y la salida tienen la relación uno a uno, y la estructura de partición del RDD resultante permanece sin cambios, por ejemplo, map.

  La entrada y la salida tienen la relación uno-a-muchos, y la estructura de partición del resultado RDD permanece sin cambios, por ejemplo, flatMap (un elemento se convierte en una secuencia que contiene varios elementos después del mapa y, a continuación, se aplana a varios elementos).

  La entrada y la salida tienen la relación uno a uno, pero la estructura de partición del resultado RDD cambia, por ejemplo, union (dos RDD se integran a un RDD, y el número de particiones se convierte en la suma del número de particiones de dos RDD) y coalescen (las particiones se reducen).

  Los operadores de algunos elementos se seleccionan de la entrada, tales como filter, distinct (los elementos duplicados se eliminan), subtract (los elementos solo existen en este RDD se conservan) y sample (se toman muestras).

- Los elementos RDD se consideran pares de clave-valor.

  Realizar el cálculo uno a uno en el único RDD, como mapValues (se conserva el modo de partición del RDD de origen, que es diferente del map).

  Ordenar el único RDD, como sort y partitionBy (partición con coherencia, lo que es importante para la optimización local).

  Reestructurar y reduce el RDD único basado en clave, como groupByKey y reduceByKey.

  Join y reestructurar dos RDD basados en la clave, como join y cogroup.

  > NOTA
  >
  > Las tres operaciones posteriores que implican la clasificación se denominan operaciones de shuffle.

Action incluye los siguientes tipos:

- Genere elementos de configuración escalar, como **count** (el número de elementos en el RDD devuelto), **reduce**, **fold/aggregate** (el número de elementos de configuración escalar que se devuelven) y **take** (el número de elementos antes del retorno).
- Generar la colección Scala, como **collect** (importar todos los elementos del RDD a la colección Scala) y **lookup** (buscar todos los valores corresponden a la clave).
- Escribir datos en el almacenamiento, como **saveAsTextFile** (que corresponde al **textFile** anterior).

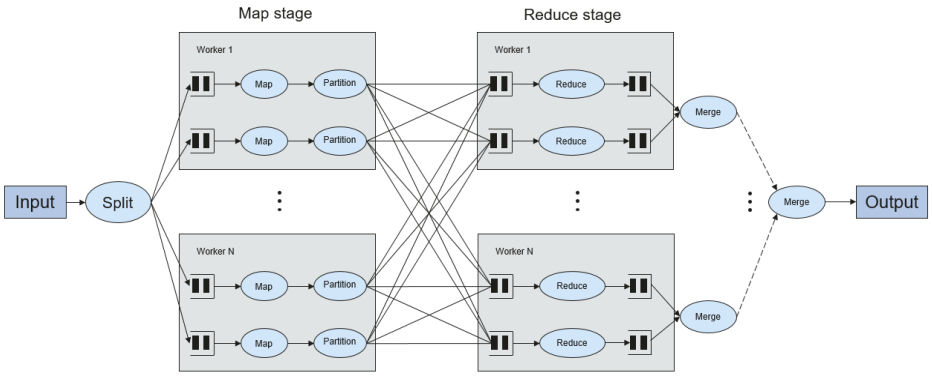
- Puntos de comprobación, como el operador **checkpoint**. Cuando el lineage es bastante largo (lo que ocurre con frecuencia en el cálculo de gráficos), se necesita un largo período de tiempo para ejecutar toda la secuencia de nuevo cuando se produce un falla. En este caso, checkpoint se utiliza como el punto de control para escribir los datos actuales en el almacenamiento estable.

- **Shuffle**

  La mezcla aleatoria es una fase específica en el marco de MapReduce, que se encuentra entre la fase Map y la fase Reduce. Si los resultados de salida de Map van a ser utilizados por Reduce, los resultados de salida deben ser hashed basados en una clave y distribuidos a cada Reducer. Este proceso se llama Shuffle. Shuffle implica la lectura y escritura del disco y la transmisión de la red, de modo que el rendimiento de Shuffle afecta directamente a la eficiencia de operación de todo el programa.

  La siguiente figura muestra todo el proceso del algoritmo de MapReduce.

  Figura 6-95 Proceso de algoritmo

  Shuffle es un puente para conectar datos. A continuación se describe la implementación de shuffle en Spark.

  Shuffle divide un job de Spark en múltiples stages. Las primeras etapas contienen uno o más ShuffleMapTasks y la última etapa contiene uno o más ResultTasks.

- **Estructura de Spark Application**

  La estructura de aplicación Spark incluye el SparkContext inicializado y el programa principal.

  - SparkContext inicializado: construye el entorno operativo de la aplicación Spark.

    Construye el objeto de SparkContext. A continuación se presenta un ejemplo:

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    Descripción de parámetros:

    master indica la cadena de enlace. Los modos de enlace incluyen local, Yarn-cluster y Yarn-cliente.

    appName indica el nombre de la aplicación.

    SparkHome indica el directorio donde está instalado Spark en el clúster.

    jars indica el código y el paquete de dependencias de una aplicación.

  - Programa principal: procesa datos.

  Para obtener más información sobre cómo presentar una solicitud, visite **https://spark.apache.org/docs/3.1.1/submitting-applications.html**.

- **Comandos de Spark Shell**

  Los comandos básicos de Spark shell soportan el envío de aplicaciones de Spark. Los comandos de shell Spark son los siguientes:

  ```
  ./bin/spark-submit \
    --class <main-class> \
    --master <master-url> \
    ... # other options
    <application-jar> \
    [application-arguments]
  ```

  Descripción de parámetros:

  --class: indica el nombre de la clase de un Spark application.

  --master: indica el patrón al que se vincula la aplicación de Spark, como Yarn-client y Yarn-cluster.

  application-jar: indica la ruta del archivo JAR de Spark application.

  application-arguments: indica el parámetro necesario para enviar Spark application. Este parámetro se puede dejar en blanco.

- **Spark JobHistory Server**

  La interfaz de usuario web de Spark se utiliza para monitorear los detalles en cada fase de marco de Spark de un trabajo de Spark en ejecución o histórico y proporcionar la visualización del registro, lo que ayuda a los usuarios a desarrollar, configurar y optimizar el trabajo en unidades más finas.

6.27.2 Solución de Spark HA

Solución de implementación y principios de HA de instancia multiactiva de Spark

Basado en JDBCServer existente en la comunidad, el modo de instancia multiactiva se utiliza para lograr HA. En este modo, varios JDBCServers coexisten en el clúster y el cliente puede conectar aleatoriamente cualquier JDBCServer para realizar operaciones de servicio. Cuando uno o varios JDBCServers dejan de funcionar, un cliente puede conectarse a otro JDBCServer normal.

En comparación con el modo HA activo/en espera, el modo de instancia multiactivo tiene las siguientes ventajas:

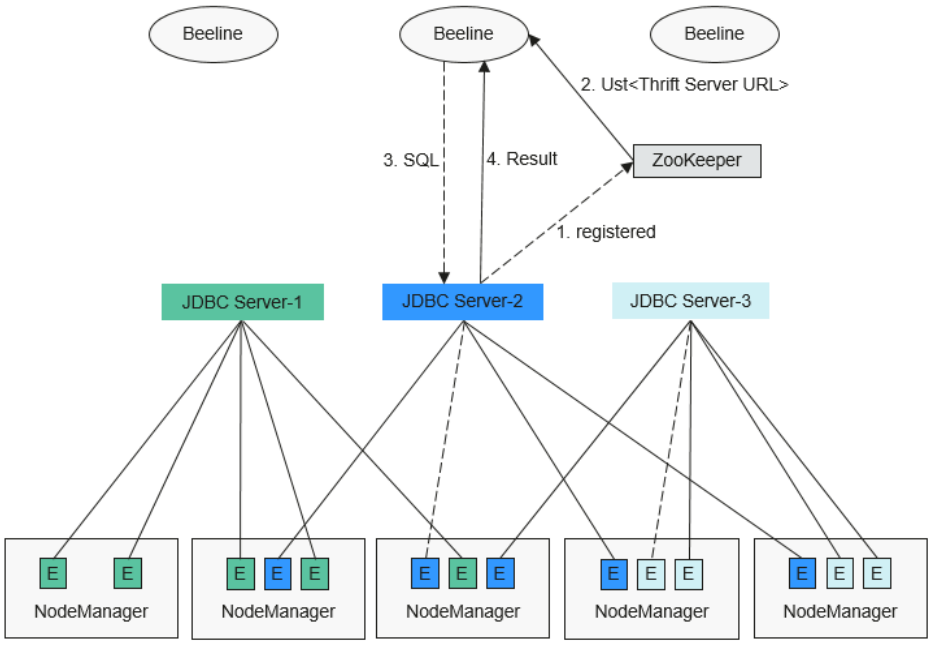
- En HA activo/en espera, cuando se produce la conmutación activa/en espera, JDBCServer no puede controlar el período no disponible, pero depende de los recursos de servicio de Yarn.
- En Spark, el Thrift JDBC similar a HiveServer2 proporciona servicios y los usuarios acceden a los servicios a través de Beeline y JDBC API. Por lo tanto, la capacidad de procesamiento del clúster de JDBCServer depende de la capacidad de punto único del servidor primario, y la escalabilidad es insuficiente.

El modo HA de instancia multiactiva no solo puede evitar la interrupción del servicio causada por la conmutación, sino que también permite el escalado horizontal del clúster para mejorar la alta concurrencia.

- **Implementación**

  La siguiente figura muestra el principio básico de HA de instancia multiactiva de Spark JDBCServer.

Figura 6-96 Spark JDBCServer HA



1. Cuando se inicia un JDBCServer, se registra con ZooKeeper escribiendo información de nodo en un directorio especificado. La información del nodo incluye la dirección IP de la instancia, el número de puerto, la versión y el número de serie.
2. Para conectarse a JDBCServer, el cliente debe especificar el espacio de nombres, que es el directorio de las instancias de JDBCServer en ZooKeeper. Durante la conexión, se selecciona aleatoriamente una instancia de JDBCServer del espacio de nombres especificado.
3. Una vez que la conexión se realiza correctamente, el cliente envía sentencias de SQL a JDBCServer.
4. JDBCServer ejecuta sentencias de SQL recibidas y devuelve los resultados al cliente.

Si el HA de instancia multiactiva de Spark JDBCServer está habilitado, todas las instancias de JDBCServer son independientes y equivalentes. Cuando se interrumpe una instancia de JDBCServer durante la actualización, otras instancias de JDBCServer pueden aceptar la solicitud de conexión del cliente.

Las siguientes reglas deben seguirse en la instancia multiactiva HA de Spark JDBCServer.

- Si una instancia de JDBCServer sale de forma anormal, ninguna otra instancia se hará cargo de las sesiones y servicios que se ejecutan en la instancia anormal.
- Cuando se detiene el proceso de JDBCServer, los nodos correspondientes se eliminan de ZooKeeper.
- El cliente selecciona aleatoriamente el servidor, lo que puede dar como resultado una asignación de sesión desigual causada por la distribución aleatoria de los resultados de la política, y finalmente dar como resultado un desequilibrio de carga de las instancias.
- Después de que la instancia entra en el modo de mantenimiento (en el que no se aceptan nuevas solicitudes de conexión de los clientes), los servicios que se ejecutan en la instancia pueden fallar cuando se agota el tiempo de desmantelamiento.
- **Conexión de URL**
  - Modo de instancia multiactiva

En el modo de instancia multiactiva, el cliente lee el contenido del nodo ZooKeeper y se conecta a JDBCServer. Las cadenas de conexión se enumeran a continuación.

- Modo de seguridad:

  Si la autenticación de Kinit está habilitada, JDBCURL es la siguiente:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
  <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
  rkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/
  hadoop.<System domain name>@<System domain name>;
  ```

  NOTA

  - En el JDBCURL anterior, **<zkNode_IP>:<zkNode_Port>** indica la URL ZooKeeper. Utilice comas (,) para separar varias URL,

    Ejemplo: 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
  - **sparkthriftserver2x** indica el directorio de ZooKeeper donde una instancia de JDBCServer aleatoria está conectada al cliente.

  Por ejemplo, cuando utiliza el cliente Beeline para conectar JDBCServer, ejecute el siguiente comando:

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.**<*System domain name*>**@**<*System domain name*>**;"**

  Si la autenticación de Keytab está habilitada, JDBCURL es la siguiente:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
  <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
  rkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/
  hadoop.<System domain name>@<System domain
  name>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
  ```

  En la URL anterior <*principal_name*> indica el principal del usuario de Kerberos, por ejemplo, **test@**<*System domain name*>; <*path_to_keytab*> indica la ruta del archivo de Keytab correspondiente a <*principal_name*>, por ejemplo, **/opt/auth/test/user.keytab**.

- Modo común:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
  <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
  rkthriftserver2x;
  ```

  Por ejemplo, cuando utiliza el cliente de Beeline, en modo normal, para la conexión, ejecute el siguiente comando:

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;"**

- Modo de instancia no multiactiva

  En este modo, un cliente se conecta a un nodo de JDBCServer especificado. En comparación con el modo de instancia multiactivo, la cadena de conexión en este modo no contiene los parámetros **serviceDiscoveryMode** y **zooKeeperNamespace** sobre ZooKeeper.

Por ejemplo, cuando utiliza el cliente de Beeline, en modo de seguridad, para conectar JDBCServer en modo de instancia no multiactivo, ejecute el siguiente comando:

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*server_IP*>:<*server_Port*>/**;user.principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;"**

NOTA

- In the above command, **<server_IP>:<server_Port>** indicates the URL of the specified JDBCServer node.
- **CLIENT_HOME** indica la ruta del cliente.

Excepto el método de conexión, otras operaciones de la API de JDBCServer en los dos modos son las mismas. Spark JDBCServer es otra implementación de HiveServer2 en Hive. Para obtener más información sobre cómo usar JDBCServer de Spark, consulte **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

Spark Multi-Tenant HA

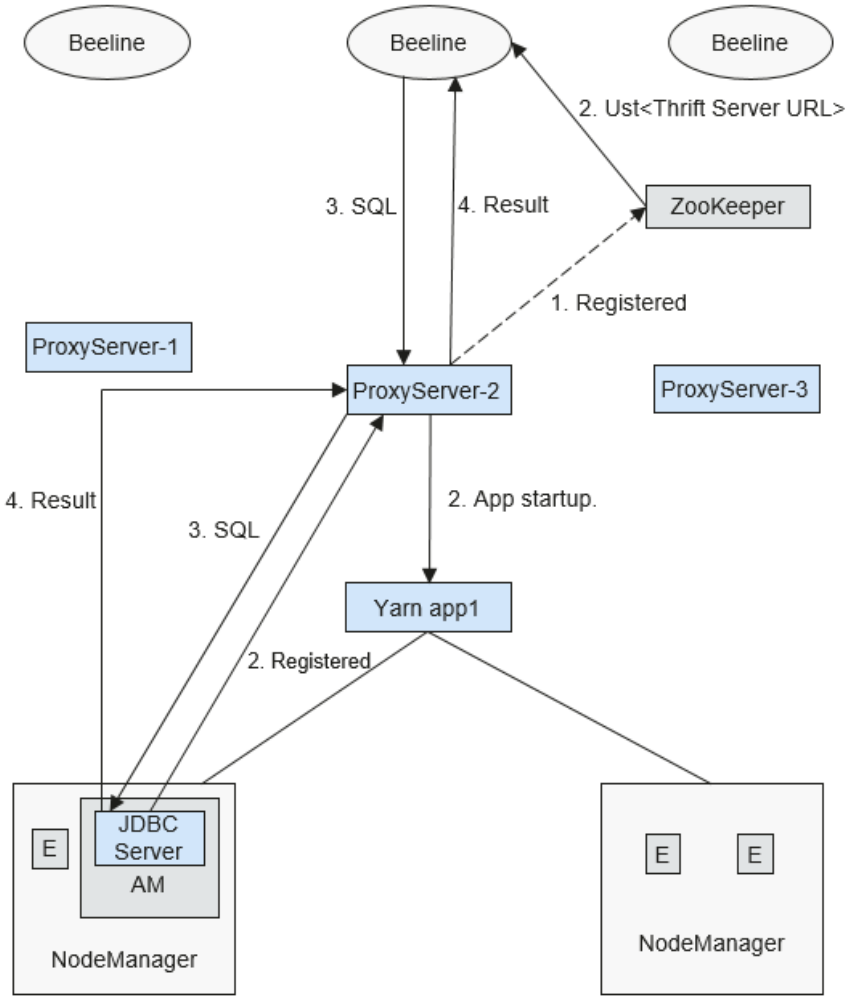
En la solución de instancia multiactiva de JDBCServer, JDBCServer utiliza el modo Yarn-client, pero solo hay una cola de recursos de Yarn disponible. Para resolver este problema de limitación de recursos, se introduce el modo multitenant.

En el modo multitenant, JDBCServers están vinculados con tenants. Cada tenant corresponde a uno o más JDBCServers, y un JDBCServer proporciona servicios para un solo tenant. Se pueden configurar diferentes tenants con diferentes colas de Yarn para implementar el aislamiento de recursos. Además, JDBCServer puede iniciarse dinámicamente según sea necesario para evitar el desperdicio de recursos.

- **Implementación**

  Figura 6-97 muestra la solución de HA del modo multitenant.

Figura 6-97 Modo multitenant de Spark JDBCServer



a. Cuando se inicia ProxyServer, se registra con ZooKeeper escribiendo información de nodo en un directorio especificado. La información del nodo incluye la dirección IP de la instancia, el número de puerto, la versión y el número de serie.

   NOTA

   En el modo multitenant, la instancia JDBCServer hace referencia al ProxyServer (JDBCServer proxy).

b. Para conectarse a ProxyServer, el cliente debe especificar un espacio de nombres, que es el directorio de la instancia de ProxyServer donde desea acceder a ZooKeeper. Cuando el cliente se conecta a ProxyServer se selecciona una instancia aleatoria bajo el espacio de nombres para la conexión. Para obtener más información sobre la dirección URL, consulte **Descripción de la conexión de URL**.

c. Después de que el cliente se conecta correctamente a ProxyServer, que primero comprueba si existe el JDBCServer de un tenant. En caso afirmativo, Beeline conecta el JDBCServer. Si no, se inicia un nuevo JDBCServer en modo de Yarn-cluster. Después del inicio de JDBCServer, ProxyServer obtiene la dirección IP del JDBCServer y establece la conexión entre Beeline y JDBCServer.

d. El cliente envía sentencias SQL a ProxyServer que reenvía las sentencias al JDBCServer conectado. JDBCServer devuelve los resultados a ProxyServer que luego devuelve los resultados al cliente.

En el modo HA de instancia multiactiva, todas las instancias son independientes y equivalentes. Si una instancia se interrumpe durante la actualización, otras instancias pueden aceptar la solicitud de conexión del cliente.

- **Descripción de conexión de URL**
  - Modo multitenant

    En modo multitenant, el cliente lee el contenido del nodo de ZooKeeper y se conecta a ProxyServer. Las cadenas de conexión se enumeran a continuación.

    - Modo de seguridad:

      Si la autenticación de Kinit está habilitada, la URL del cliente es la siguiente:

      ```
      jdbc:hive2://
      <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
      <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
      rkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/
      hadoop.<System domain name>@<System domain name>;
      ```

      NOTA

      - En la URL anterior, **<zkNode_IP>:<zkNode_Port>** indica la URL de ZooKeeper. Utilice comas (,) para separar varias URL,

        Ejemplo: **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
      - **sparkthriftserver2x** indica el directorio de ZooKeeper donde una instancia de JDBCServer aleatoria está conectada al cliente.

      Por ejemplo, cuando utiliza el cliente de Beeline para la conexión, ejecute el siguiente comando:

      sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.**<*System domain name*>**@**<*System domain name*>**;"**

      Si la autenticación de Keytab está habilitada, la URL es la siguiente:

      ```
      jdbc:hive2://
      <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
      <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
      rkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/
      hadoop.<System domain name>@<System domain
      name>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
      ```

      En la URL anterior <*principal_name*> indica el principal del usuario de Kerberos, por ejemplo, **test@**<*System domain name*>; <*path_to_keytab*> indica la ruta del archivo de Keytab correspondiente a <*principal_name*>, por ejemplo, **/opt/auth/test/user.keytab**.

    - Modo común:

      ```
      jdbc:hive2://
      <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
      <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
      rkthriftserver2x;
      ```

      Por ejemplo, ejecute el siguiente comando cuando utilice el cliente de Beeline para la conexión en modo normal:

      sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode*

3_IP>:<zkNode3_Port>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;"**

- Modo no multitenant

  En modo no multitenant, un cliente se conecta a un nodo de JDBCServer especificado. En comparación con el modo de instancia multi-inquilino, la cadena de conexión en este modo no contiene los parámetros **serviceDiscoveryMode** y **zooKeeperNamespace** sobre ZooKeeper.

  Por ejemplo, cuando utiliza el cliente de Beeline para conectar JDBCServer en modo de instancia de non-multitenant, ejecute el siguiente comando:

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*server_IP*>:<*server_Port*>/**;user.principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;"**

  NOTA

  - En el comando anterior, **<server_IP>:<server_Port>** indica la URL del nodo JDBCServer especificado.
  - **CLIENT_HOME** indica la ruta del cliente.

  Excepto el método de conexión, otras operaciones de JDBCServer API en modo de multitenant y modo non-multitenant son las mismas. Spark JDBCServer es otra implementación de HiveServer2 en Hive. Para obtener más información sobre cómo usar Spark JDBCServer, visite el sitio web oficial de Hive en **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

  Especificación de un tenant

  Generalmente, el cliente enviado por un usuario se conecta al JDBCServer predeterminado del tenant al que pertenece el usuario. Si desea conectar el cliente al JDBCServer de un tenant especificado, agregue el parámetro **--hiveconf mapreduce.job.queuename**.

  Si utiliza el cliente Beeline para la conexión, ejecute el siguiente comando (el nombre del tenant es **aaa**):

  beeline --hiveconf mapreduce.job.queuename=aaa -u 'jdbc:hive2://192.168.39.30:2181,192.168.40.210:2181,192.168.215.97:2181;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.<*System domain name*>@<*System domain name*>;'

6.27.3 Relación entre Spark, HDFS y Yarn

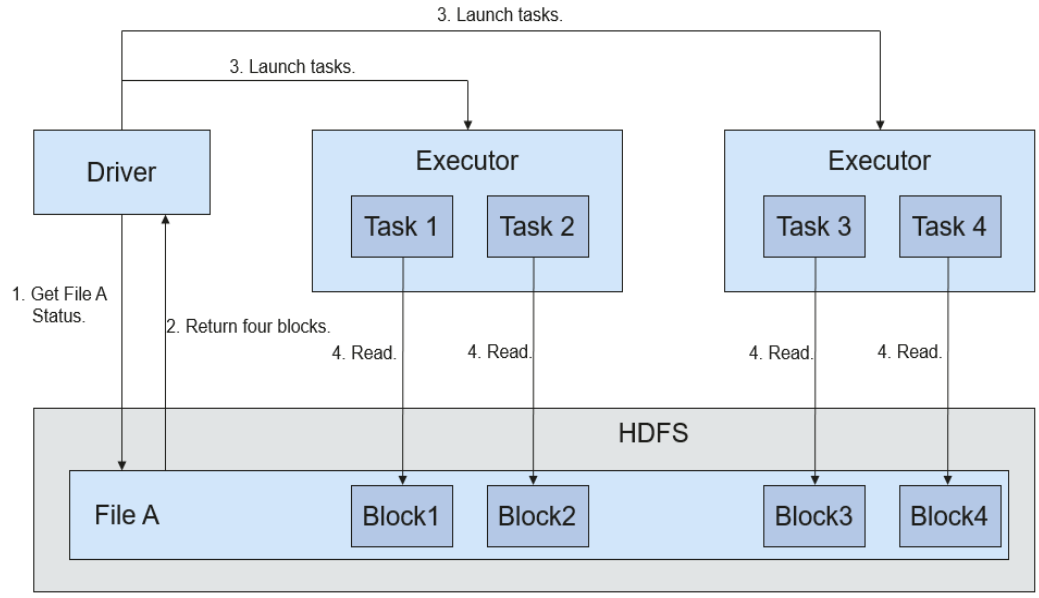
Relación entre Spark y HDFS

Los datos calculados por Spark provienen de múltiples fuentes de datos, como archivos locales y HDFS. La mayoría de los datos calculados por Spark provienen del HDFS. El HDFS puede leer datos a gran escala para computación paralela. Después de ser calculados, los datos se pueden almacenar en el HDFS.

Spark implica Driver y Executor. El Driver programa las tareas y el Executor ejecuta las tareas.

Figura 6-98 muestra el proceso de lectura de un archivo.

Figura 6-98 Proceso de lectura de archivos

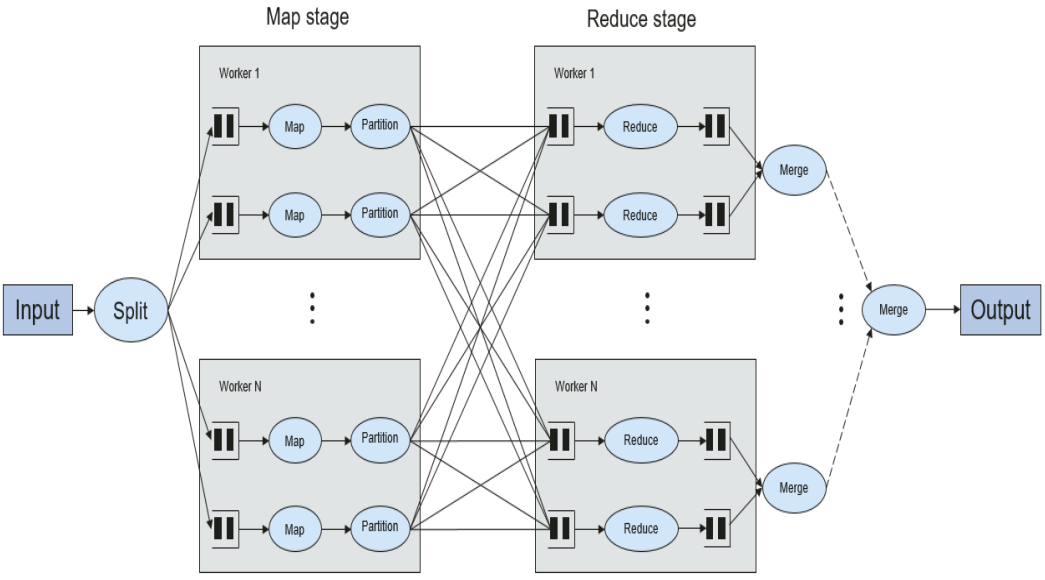


El proceso de lectura de archivos es el siguiente:

1. Driver se interconecta con el HDFS para obtener la información del File A.
2. El HDFS devuelve la información de block detallada acerca de este archivo.
3. Driver establece un grado paralelo basado en la cantidad de datos de block y crea varias tasks para leer los blocks de este archivo.
4. Executor ejecuta las tareas y lee los blocks detallados como parte del conjunto de datos distribuido resistentes (RDD).

Figura 6-99 muestra el proceso de escribir datos en un archivo.

Figura 6-99 Proceso de escritura de archivos



El proceso de escritura de archivos es el siguiente:

1. Driver crea un directorio donde se va a escribir el archivo.
2. Basándose en el estado de distribución de RDD, se calcula el número de tasks relacionadas con la escritura de datos, y estas tareas se envían al Executor.
3. Executor ejecuta estas tasks y escribe los datos RDD en el directorio creado en **1**.

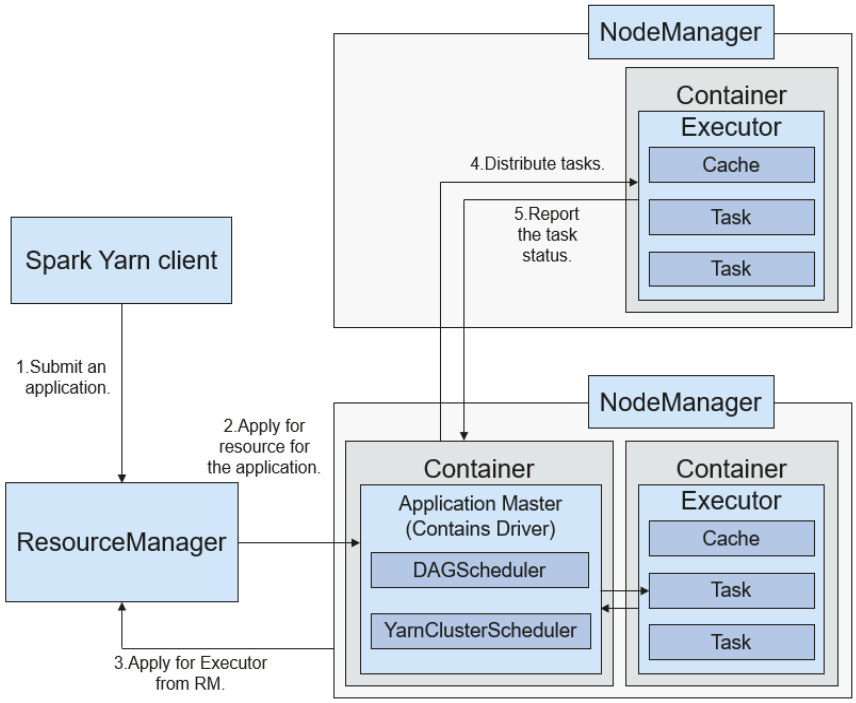
Relación entre Spark y Yarn

La computación y programación de Spark pueden implementarse usando el modo de Yarn. Spark disfruta de los recursos informáticos proporcionados por los clústeres de Yarn y ejecuta tareas de forma distribuida. Spark en Yarn tiene dos modos: Yarn-cluster y Yarn-client.

- Modo de Yarn-cluster

  Figura 6-100 muestra el marco de ejecución de Spark en Yarn-cluster.

  Figura 6-100 Marco de operación de Spark on Yarn-cluster



  Proceso de implementación de Spark on Yarn-cluster:

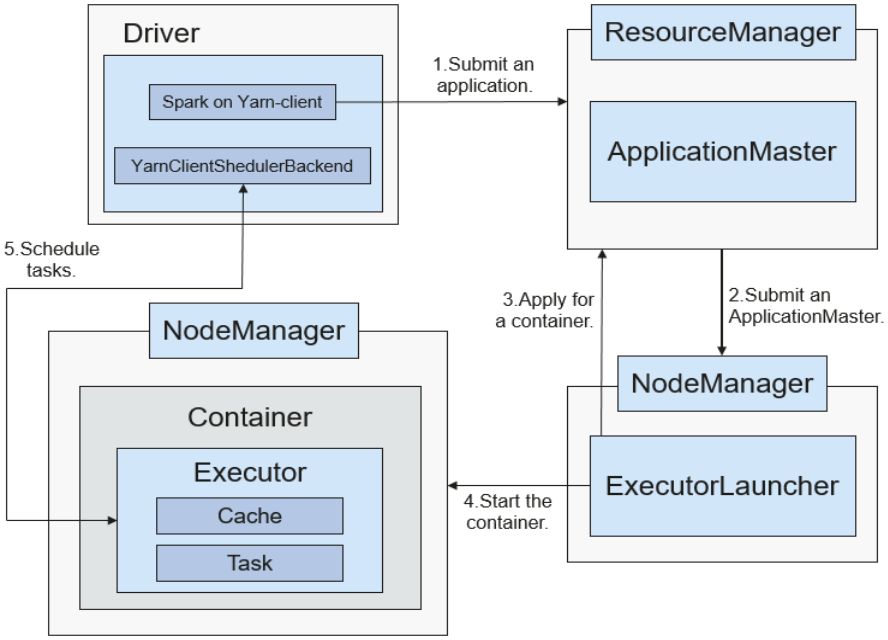
  a. El cliente genera la información de la aplicación y, a continuación, envía la información a ResourceManager.
  b. ResourceManager asigna el primer container (ApplicationMaster) a SparkApplication e inicia el driver en el container.
  c. ApplicationMaster se aplica a los recursos de ResourceManager para ejecutar el container.

     ResourceManager asigna el container a ApplicationMaster, que se comunica con NodeManager, e inicia el executor en el container obtenido. Después de iniciar el executor, se registra con el driver y se aplica a tasks.
  d. El driver asigna tasks al executor.
  e. El executor ejecuta tasks e informa del estado operativo al driver.

- Modo de Yarn-client

  Figura 6-101 muestra el marco de ejecución de Spark en Yarn-cluster.

  Figura 6-101 Marco de operación de Spark on Yarn-client



  Proceso de implementación de Spark on Yarn-client:

  NOTA

  En el modo Yarn-client, Driver se despliega en el client y se inicia en el client. En el modo Yarn-client, el cliente de la versión anterior es incompatible. Se recomienda utilizar el modo Yarn-cluster.

  a. El cliente envía la solicitud de aplicación de Spark a ResourceManager y, a continuación, ResourceManager devuelve los resultados. Los resultados incluyen información como Application ID y los recursos máximos y mínimos disponibles. El cliente empaqueta toda la información necesaria para iniciar ApplicationMaster y envía la información a ResourceManager.

  b. Después de recibir la solicitud, ResourceManager encuentra un nodo apropiado para ApplicationMaster y lo inicia en este nodo. ApplicationMaster es un papel en Yarn, y el nombre del proceso en Spark es ExecutorLauncher.

  c. En función de los requerimientos de recursos de cada tarea, el ApplicationMaster puede solicitar una serie de containers para ejecutar tareas desde ResourceManager.

  d. Después de recibir la lista de container recién asignados (de ResourceManager), ApplicationMaster envía información a los NodeManagers relacionados para iniciar los containers.

     ResourceManager asigna los containers a ApplicationMaster, que se comunica con los NodeManagers relacionados e inicia los executors en los containers obtenidos. Una vez iniciados los executors, se registra con los drivers y se aplica a tasks.

     NOTA

     Los containers en ejecución no se suspenden y los recursos no se liberan.

e. Los drivers asignan tasks a los executors. El executor ejecuta tasks e informa del estado operativo al driver.

6.27.4 Función de código abierto mejorado de Spark: consulta de SQL optimizada de datos de origen cruzado

Escenario

Las empresas suelen almacenar datos masivos, como de varias bases de datos y almacenes, para la gestión y la recopilación de información. Sin embargo, las fuentes de datos diversificadas, las estructuras de conjuntos de datos híbridos y el almacenamiento de datos dispersos reducen la eficiencia de las consultas.

El Spark de código abierto solo admite un simple pushdown de filtro durante la consulta de datos de múltiples fuentes. El rendimiento del motor SQL se deteriora debido a una gran cantidad de transmisión de datos innecesaria. La función pushdown se mejora, de modo que **aggregate**, **projection** complejo, y**predicate** complejo se pueden enviar a las fuentes de datos, reduciendo la transmisión de datos innecesaria y mejorando el rendimiento de la consulta.

Solo el origen de datos JDBC admite operaciones de consulta, como **aggregate**, **projection**, **predicate**, **aggregate over inner join** y **aggregate over union all**. Todas las operaciones de pushdown se pueden habilitar en función de sus necesidades.

Tabla 6-23 Consulta mejorada de consulta entre fuentes

Módulo	Antes de la mejora	Después de la mejora
aggregate	El pushdown de **aggregate** no es compatible.	● Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. Ejemplo: select count(*) from table ● Se admiten expresiones internas de funciones de agregación. Ejemplo: select sum(a+b) from table ● Se admite el cálculo de funciones de agregación. Ejemplo: select avg(a) + max(b) from table ● Se admite pushdown de **having**. Ejemplo: select sum(a) from table where a>0 group by b having sum(a)>10 ● Se soporta el pushdown de algunas funciones. Se admite la reducción de líneas en matemáticas, tiempo y funciones de cadena, como **abs()**, **month()** y **length()**. Además de las funciones integradas anteriores, puede ejecutar el comando **SET** para agregar funciones compatibles con los orígenes de datos. Ejemplo: select sum(abs(a)) from table ● Se admite pushdown de **limit** y **order by** después de **aggregate**. Sin embargo, Oracle no admite el pushdown, ya que Oracle no admite **limit**. Ejemplo: select sum(a) from table where a>0 group by b order by sum(a) limit 5
projection	Solo se soporta el pushdown de **projection** simple. Ejemplo: select a, b from table	● Las expresiones complejas pueden ser presionadas hacia abajo. Ejemplo: select (a+b)*c from table ● Algunas funciones pueden ser presionadas hacia abajo. Para obtener más información, consulte la descripción que se encuentra debajo de la tabla. Ejemplo: select length(a)+abs(b) from table ● Se admite pushdown de **limit** y **order by** después de **projection**. Ejemplo: select a, b+c from table order by a limit 3

Módulo	Antes de la mejora	Después de la mejora
predicate	Solo se admite el filtrado simple con el nombre de columna a la izquierda del operador y los valores a la derecha. Ejemplo: seleccione * de la tabla donde a>0 o b en ("aaa", "bbb")	● Se admite el pushdown de expresiones complejas. Ejemplo: select * from table where a +b>c*d or a/c in (1, 2, 3) ● Algunas funciones pueden ser presionadas hacia abajo. Para obtener más información, consulte la descripción que se encuentra debajo de la tabla. Ejemplo: select * from table where length(a)>5
aggregate over inner join	Los datos relacionados de las dos tablas deben cargarse en Spark. La operación de join debe realizarse antes de la operación **aggregate**.	Se soportan las siguientes funciones: ● Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. ● Todas las operaciones de **aggregate** se pueden realizar en una misma tabla. Las operaciones de **group by** se pueden realizar en una o dos tablas y solo se admite el inner join. No se admiten los siguientes escenarios: ● **aggregate** no se puede empujar hacia abajo desde las tablas de join izquierda y derecha. ● **aggregate** contiene operaciones, por ejemplo, sum (a+b). ● Operaciones de **aggregate**, por ejemplo, sum(a)+min(b).
aggregate over union all	Los datos relacionados de las dos tablas deben cargarse en Spark. **union** debe realizarse antes de **aggregate**.	Escenarios admitidos: Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. Escenarios no admitidos: ● **aggregate** contiene operaciones, por ejemplo, sum (a+b). ● Operaciones de **aggregate**, por ejemplo, sum(a)+min(b).

Precauciones

- Si el origen de datos externo es Hive, la operación de consulta no se puede realizar en tablas externas creadas por Spark.
- Solo se admiten las fuentes de datos MySQL y MPPDB.

6.28 Spark2x

6.28.1 Principios básicos de Spark2x

NOTA

El componente de Spark2x se aplica a MRS 3.x y versiones posteriores.

Descripción

Spark es un marco de computación distribuida basado en memoria. En escenarios de computación iterativos, la capacidad de computación de Spark es de 10 a 100 veces mayor que MapReduce porque los datos se almacenan en memoria cuando se procesan. Spark puede utilizar HDFS como sistema de almacenamiento subyacente, lo que permite a los usuarios cambiar rápidamente a Spark desde MapReduce. Spark ofrece capacidades de análisis de datos integrales, como el procesamiento de streaming en lotes pequeños, procesamiento por lotes sin conexión, consultas de SQL y minería de datos. Los usuarios pueden utilizar estas funciones sin problemas en una misma aplicación. Para obtener más información sobre las nuevas características de código abierto de Spark2x, consulte **Nuevas funciones de código abierto de Spark2x**.

Las características de Spark son las siguientes:

- Mejora la capacidad de procesamiento de datos a través de la computación de memoria distribuida y el motor de ejecución de gráficos acíclicos dirigidos (DAG). El rendimiento entregado es de 10 a 100 veces mayor que el de MapReduce.
- Soporta múltiples lenguajes de desarrollo (Scala/Java/Python) y docenas de operadores altamente abstractos para facilitar la construcción de aplicaciones de procesamiento de datos distribuidos.
- Crea pilas de procesamiento de datos usando **SQL**, **Streaming**, MLlib y GraphX para proporcionar capacidades de procesamiento de datos de una sola parada.
- Se adapta al ecosistema de Hadoop, lo que permite que las aplicaciones de Spark se ejecuten en Standalone, Mesos o Yarn, lo que permite el acceso a múltiples fuentes de datos como HDFS, HBase y Hive, y admite la migración sin problemas de la aplicación MapReduce a Spark.

Arquitectura

Figura 6-102 describe la arquitectura de Spark y **Tabla 6-24** enumera los módulos de Spark.

Figura 6-102 Arquitectura de Spark

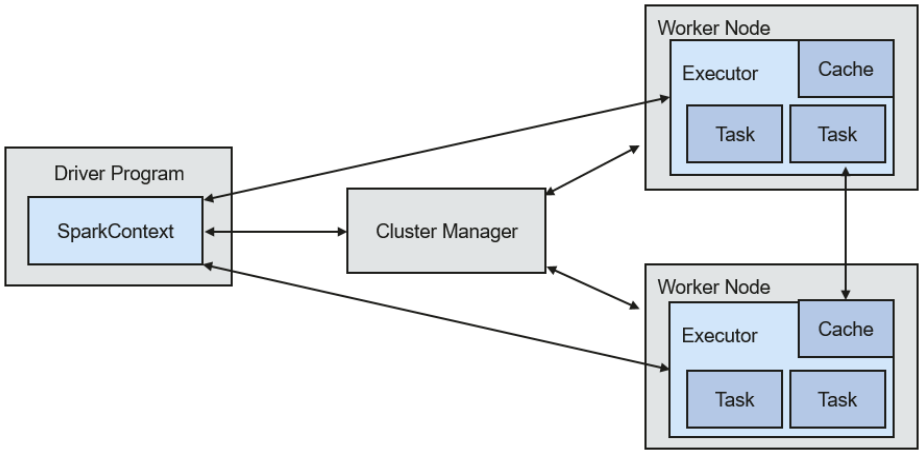


Tabla 6-24 Conceptos básicos

Módulo	Descripción
Cluster Manager	El administrador de clústeres gestiona los recursos del clúster. Spark admite varios administradores de clústeres, incluidos Mesos, Yarn y el administrador de clústeres independiente que se entrega con Spark. De forma predeterminada, los clústeres de Spark adoptan el administrador de clústeres de Yarn.
Application	Aplicación de Spark. Consiste en un programa de Driver Program y múltiples executors.
Deploy Mode	Implementación en modo de clúster o cliente. En modo de clúster, el controlador se ejecuta en un nodo dentro del clúster. En modo cliente, el controlador se ejecuta en el cliente (fuera del clúster).
Driver Program	El proceso principal de la aplicación Spark. Ejecuta la función **main()** de una aplicación y crea "SparkContext". Se utiliza para analizar aplicaciones, generar etapas y programar tareas a ejecutores. Por lo general, SparkContext representa Driver Program.
Executor	Un proceso iniciado en un Work Node. Se utiliza para ejecutar tareas, y gestionar y procesar los datos utilizados en las aplicaciones. Una aplicación de Spark generalmente contiene varios executors. Cada executor recibe comandos del driver y ejecuta una o varias tareas.
Worker Node	Nodo que inicia y gestiona ejecutores y recursos en un clúster.
Job	Un job consta de varias tareas simultáneas. Un operador de action (por ejemplo, un operador de collect) se asigna a un job.
Stage	Cada job consta de múltiples etapas. Cada etapa es un conjunto de tareas, que está separado por el Gráfico Acíclico Dirigido (DAG).

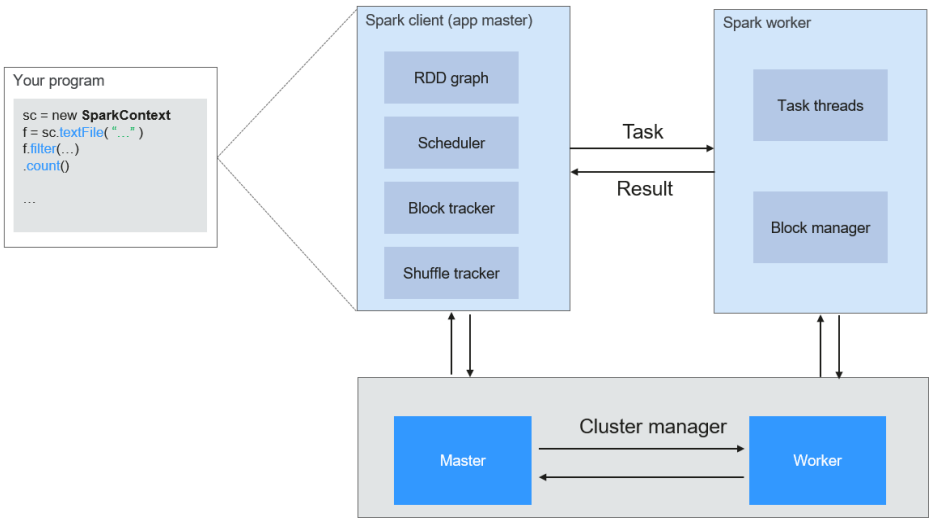
Módulo	Descripción
Task	Una tarea lleva la unidad de cálculo de la lógica de servicio. Es la unidad de trabajo mínima que se puede ejecutar en la plataforma de Spark. Una aplicación se puede dividir en múltiples tareas basadas en el plan de ejecución y la cantidad de cálculo.

Principio de Spark

Figura 6-103 describe la arquitectura de Spark en ejecución de aplicaciones.

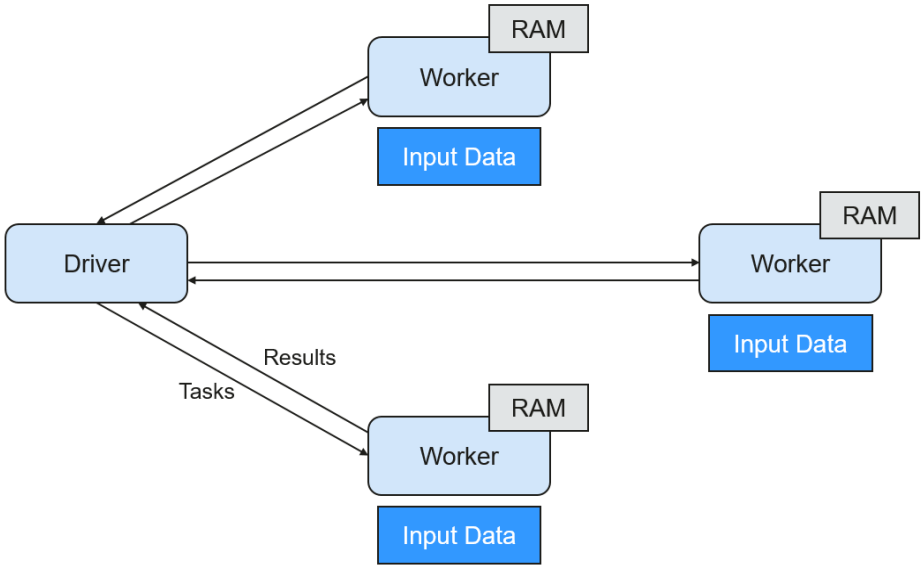
1. Una aplicación se está ejecutando en el clúster como una colección de procesos. El controlador coordina la ejecución de la aplicación.
2. Para ejecutar una aplicación, el controlador se conecta al administrador de clústeres (como Standalone, Mesos y Yarn) para solicitar los recursos del ejecutor e iniciar ExecutorBackend. El administrador de clúster programa los recursos entre diferentes aplicaciones. El controlador programa los DAGs, divide las etapas y genera tareas para la aplicación al mismo tiempo.
3. A continuación, Spark envía los códigos de la aplicación (los códigos transferidos a **SparkContext** definidos por JAR o Python) a executor.
4. Una vez terminadas todas las tareas, se detiene la ejecución de la aplicación de usuario.

Figura 6-103 Arquitectura de ejecución de aplicaciones de Spark



Spark utiliza los modos de Master y Worker, como se muestra en **Figura 6-104**. Un usuario envía una aplicación en el cliente de Spark y, a continuación, el programador divide un job en varias tareas y envía las tareas a cada Worker para su ejecución. Cada Worker informa los resultados del cálculo al Driver (Master) y, a continuación, el Driver agrega y devuelve los resultados al cliente.

Figura 6-104 Modo de Spark Master-Worker



Tenga en cuenta lo siguiente acerca de la arquitectura:

- Las aplicaciones se aíslan entre sí.

  Cada aplicación tiene un proceso de executor independiente, y cada executor inicia varios subprocesos para ejecutar tareas en paralelo. Cada driver programa sus propias tareas, y diferentes tareas de aplicación se ejecutan en diferentes JVM, es decir, diferentes executors.
- Las diferentes aplicaciones de Spark no comparten datos, a menos que los datos se almacenen en el sistema de almacenamiento externo, como HDFS.
- Se recomienda desplegar el programa de Driver en una ubicación cercana al nodo de Worker porque el programa de Driver programa tareas en el clúster. Por ejemplo, desplegar el programa de Driver en la red donde se encuentra el nodo de Worker.

Spark en YARN se puede desplegar en dos modos:

- En el modo de Yarn-cluster, el driver de Spark se ejecuta dentro de un proceso de ApplicationMaster que es getionado por Yarn en el clúster. Después de iniciar el ApplicationMaster, el cliente puede salir sin interrumpir la ejecución del servicio.
- En el modo de Yarn-client, el Driver se ejecuta en el proceso de cliente y el proceso de ApplicationMaster solo se utiliza para solicitar recursos de Yarn.

Principio de Spark Streaming

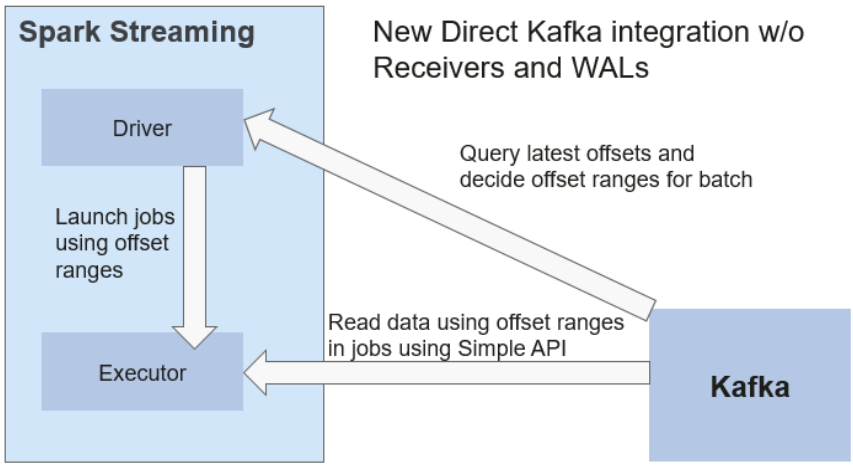
Spark Streaming es un marco informático en tiempo real basado en Spark, que amplía la capacidad para procesar datos de streaming masivos. Spark admite dos enfoques de procesamiento de datos: Direct Streaming y Receiver.

Proceso de computación de Direct Streaming

En el enfoque de Direct Streaming, Direct API se utiliza para procesar datos. Tomemos Kafka Direct API como ejemplo. Direct API proporciona una ubicación de desplazamiento desde la que cada rango de batch leerá, lo que es mucho más simple que iniciar un receptor para recibir continuamente datos de Kafka y datos escritos en registros de escritura previa (WAL). A continuación, cada job de batch se está ejecutando y los datos de desplazamiento

correspondientes están listos en Kafka. Esta información de desplazamiento puede almacenarse de forma segura en el archivo de punto de control y leerse por aplicaciones que no se iniciaron.

Figura 6-105 Transmisión de datos a través de Direct Kafka API



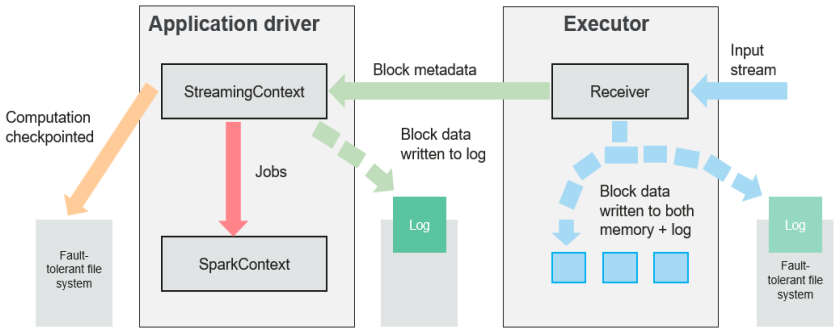
Después de la falla, Spark Streaming puede leer datos de Kafka de nuevo y procesar el segmento de datos. El resultado del procesamiento es el mismo sin importar que Spark Streaming falle o no, porque la semántica se procesa solo una vez.

Direct API no necesita usar el WAL y Receivers, y se asegura de que cada registro de Kafka se reciba solo una vez, lo que es más eficiente. De esta manera, Spark Streaming y Kafka se pueden integrar bien, haciendo que los canales de streaming se presenten con alta tolerancia a fallas, alta eficiencia y facilidad de uso. Por lo tanto, se recomienda utilizar Direct Streaming para procesar datos.

Proceso de computación de Receiver

Cuando se inicia una aplicación de Spark Streaming (es decir, cuando se inicia el driver), el StreamingContext relacionado (la base de todas las funciones de streaming) utiliza SparkContext para iniciar el receiver y convertirse en una tarea en ejecución a largo plazo. Estos receivers reciben y guardan datos de transmisió n en la memoria de Spark para su procesamiento. **Figura 6-106** muestra el ciclo de vida de la transferencia de datos.

Figura 6-106 Ciclo de vida de la transferencia de datos



1. Recibir datos (flecha azul).

Receiver divide un flujo de datos en una serie de bloques y los almacena en la memoria del executor. Además, después de habilitar WAL, escribe datos en el WAL del sistema de archivos tolerante a fallas.

2. Notificar al driver (flecha verde).

   Los metadatos en el bloque recibido se envían a StreamingContext en el driver. Los metadatos incluyen:

   - ID de referencia de bloque utilizado para localizar la posición de los datos en la memoria del Executor.
   - Información de desplazamiento de datos de bloque en registros (si la función WAL está habilitada).

3. Procesar datos (flecha roja).

   Para cada lote de datos, StreamingContext utiliza información de bloque para generar conjuntos de datos distribuidos resilientes (RDD) y jobs. StreamingContext ejecuta jobs ejecutando tareas para procesar bloques en la memoria del executor.

4. Establecer periódicamente puntos de control (flechas naranjas).
5. Para la tolerancia a fallas, StreamingContext establece periódicamente puntos de comprobación y los guarda en sistemas de archivos externos.

Tolerancia a fallas

Spark y su RDD permiten procesar sin problemas las fallas de cualquier nodo de Worker en el clúster. Spark Streaming está construido sobre Spark. Por lo tanto, el nodo Worker de Spark Streaming también tiene la misma capacidad de tolerancia a fallas. Sin embargo, Spark Streaming necesita funcionar correctamente en caso de que se ejecute durante mucho tiempo. Por lo tanto, Spark debe ser capaz de recuperarse de fallas a través del proceso del driver (proceso principal que coordina a todos los Workers). Esto plantea desafíos a la tolerancia a fallas del Spark driver debido a que el Spark driver puede ser cualquier aplicación de usuario implementada en cualquier modo de computación. Sin embargo, Spark Streaming tiene una arquitectura de computación interna. Es decir, ejecuta periódicamente el mismo computación de Spark en cada lote de datos. Dicha arquitectura le permite almacenar periódicamente puntos de control en un espacio de almacenamiento confiable y recuperarlos al reiniciar el Driver.

Para los datos de origen tales como archivos, el mecanismo de recuperación del Driver puede garantizar cero pérdida de datos porque todos los datos se almacenan en un sistema de archivos tolerante a fallas como HDFS. Sin embargo, para otras fuentes de datos tales como Kafka y Flume, algunos datos recibidos se almacenan en caché solo en la memoria y pueden perderse antes de ser procesados. Esto es causado por el modo de operación de distribución de las aplicaciones de Spark. Cuando el proceso del driver falla, todos los executors que se ejecutan en el Cluster Manager, junto con todos los datos de la memoria, se terminan. Para evitar dicha pérdida de datos, la función WAL se agrega a Spark Streaming.

WAL se utiliza a menudo en bases de datos y sistemas de archivos para garantizar la persistencia de cualquier operación de datos. Es decir, primero registrar una operación en un registro persistente y realizar esta operación en los datos. Si la operación falla, el sistema se recupera leyendo el registro y volviendo a aplicar la operación preestablecida. A continuación se describe cómo utilizar WAL para garantizar la persistencia de los datos recibidos:

Receiver se utiliza para recibir datos de fuentes de datos tales como Kafka. Como una tarea que se ejecuta durante mucho tiempo en Executor, Receiver recibe datos y también confirma los datos recibidos si son compatibles con los orígenes de datos. Los datos recibidos se almacenan en la memoria de Executor, y Driver entrega una tarea al Executor para su procesamiento.

Después de habilitar WAL, todos los datos recibidos se almacenan en archivos de registro en el sistema de archivos tolerante a fallas. Por lo tanto, los datos recibidos no pierden incluso si Spark Streaming falla. Además, el receiver comprueba la exactitud de los datos recibidos solo después de que los datos se hayan escrito previamente en registros. Los datos almacenados en caché pero no almacenados pueden ser enviados de nuevo por los orígenes de datos después de que el driver se reinicie. Estos dos mecanismos aseguran cero pérdida de datos. Es decir, todos los datos se recuperan de los registros o se reenvían por las fuentes de datos.

Para habilitar la función WAL, realice las siguientes operaciones:

- Establezca **streamingContext.checkpoint** (ruta al directorio) para configurar el directorio de checkpoint, que es una ruta de archivo de HDFS utilizada para almacenar checkpoints y WAL de streaming.
- Establezca **spark.streaming.receiver.writeAheadLog.enable** de SparkConf a **true** (el valor predeterminado es de **false**).

Después de que WAL está habilitado, todos los receivers tienen la ventaja de recuperarse de datos recibidos confiables. Se recomienda desactivar el mecanismo de réplica múltiple porque el sistema de archivos tolerante a errores de WAL también puede replicar los datos.

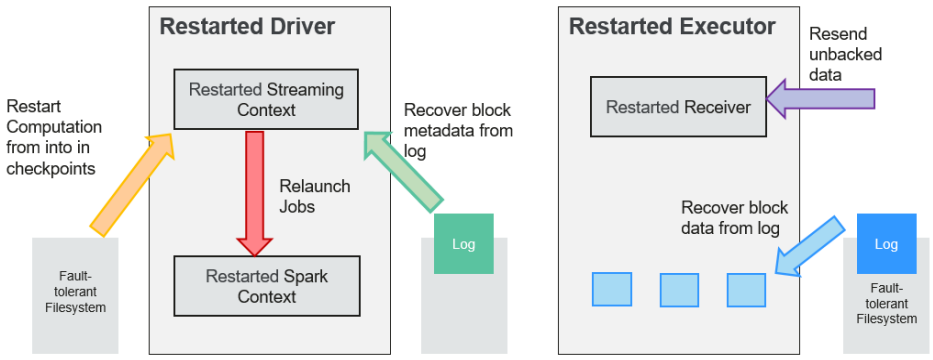
NOTA

El rendimiento de recepción de datos se reduce después de que se habilita WAL. Todos los datos se escriben en el sistema de archivos tolerante a fallas. Como resultado, el rendimiento de escritura del sistema de archivos y el ancho de banda de red para la replicación de datos puede convertirse en el cuello de botella potencial. Para resolver este problema, se recomienda crear más receptores para aumentar el grado de paralelismo de recepción de datos o utilizar mejor hardware para mejorar el rendimiento del sistema de archivos tolerante a fallas.

Proceso de recuperación

Cuando se reinicia un driver fallido, reinícielo de la siguiente manera:

Figura 6-107 Proceso de recuperación de computación



1. Recuperar la computación. (flecha naranja)

   Utilice la información del punto de control para reiniciar Driver, reconstruir el SparkContext y reinicia Receiver.

2. Recuperar el bloque de metadatos. (flecha verde)

   Esta operación asegura que todos los bloques de metadatos necesarios se recuperan para continuar con la recuperación informática posterior.

3. Relanzar trabajos inacabados. (flecha roja)

Los metadatos recuperados se utilizan para generar RDD y trabajos correspondientes para el procesamiento por lotes interrumpido debido a fallas.

4. Leer los datos del bloque guardados en los registros. (flecha azul)

   Los datos de bloque se leen directamente de los WAL durante la ejecución de los trabajos anteriores y, por lo tanto, se recuperan todos los datos esenciales almacenados de forma confiable en los registros.

5. Reenviar datos no confirmados. (flecha púrpura)

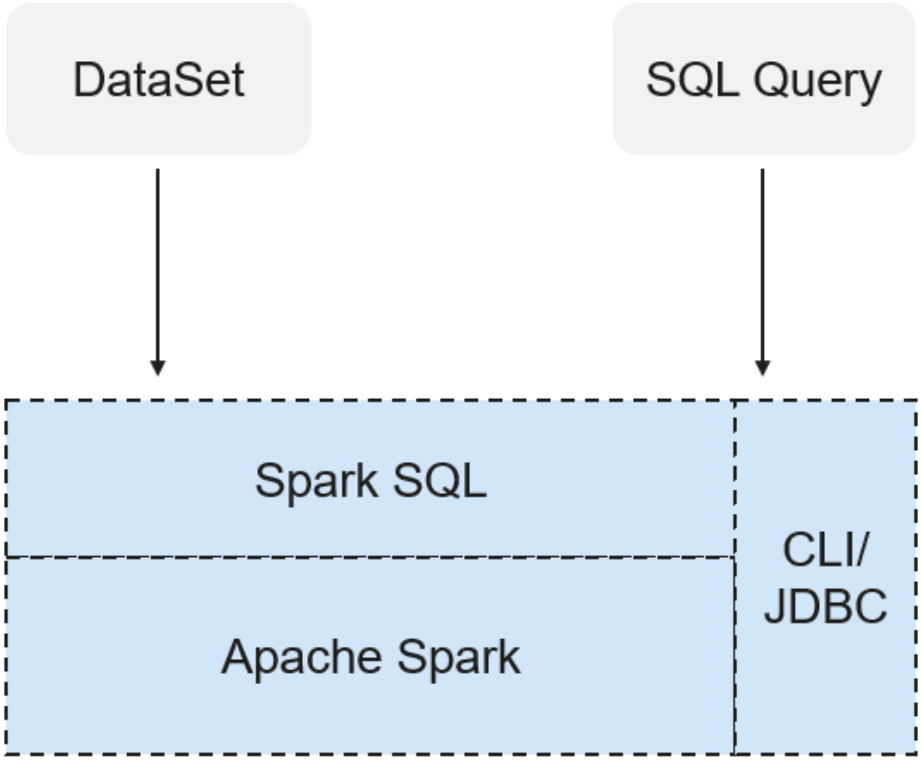
   Los datos que se almacenan en caché pero no se almacenan en los registros en caso de fallas son reenviados por las fuentes de datos, porque el receiver no confirma los datos.

Por lo tanto, mediante el uso de WALs y Receiver confiable, Spark Streaming puede evitar la pérdida de datos de entrada causada por fallas de Driver.

Principio de SparkSQL y DataSet

SparkSQL

Figura 6-108 SparkSQL y DataSet



Spark SQL es un módulo para procesar datos estructurados. En la aplicación de Spark, las sentencias de SQL o las API de DataSet se pueden usar sin problemas para consultar datos estructurados.

Spark SQL y DataSet también proporcionan un método universal para acceder a múltiples fuentes de datos como Hive, CSV, Parquet, ORC, JSON y JDBC. Estas fuentes de datos también permiten la interacción de datos. Spark SQL reutiliza la lógica de procesamiento de frontend Hive y el módulo de procesamiento de metadatos. Con Spark SQL, puede consultar directamente los datos de Hive existentes.

Además, Spark SQL también proporciona API, CLI y API JDBC, lo que permite diversos accesos al cliente.

Spark SQL Native DDL/DML

En Spark 1.5, muchos comandos de Lenguaje de definición de datos (DDL)/Lenguaje de manipulación de datos (DML) se presionan hacia abajo y se ejecutan en Hive, causando acoplamiento con Hive e inflexibilidad, como informes de errores inesperados y resultados.

Spark2x realiza la localización de comandos y reemplaza Hive con Spark SQL Native DDL/DML para ejecutar comandos de DDL/DML. Además, se realiza el desacoplamiento de Hive y se pueden personalizar los comandos.

DataSet

Un DataSet es una colección fuertemente tipada de objetos específicos de dominio que se pueden transformar en paralelo usando operaciones funcionales o relacionales. Cada Dataset también tiene una vista no escrita llamada DataFrame, que es un Dataset de Row.

El DataFrame es un conjunto de datos estructurado y distribuido que consta de varias columnas. El DataFrame es igual a una tabla en la base de datos de relaciones o al DataFrame en R/Python. El DataFrame es el concepto más básico en Spark SQL, que se puede crear mediante múltiples métodos, como el conjunto de datos estructurado, la tabla Hive, la base de datos externa o RDD.

Las operaciones disponibles en el DataSets se dividen en transformaciones y acciones.

- Una operación de transformation puede generar un nuevo DataSet,
  por ejemplo **map**, **filter**, **select** y **aggregate (groupBy)**.
- Una operación de action puede desencadenar el cálculo y devolver resultados,
  por ejemplo **count**, **show** o escribir datos en el sistema de archivos.

Puede utilizar cualquiera de los siguientes métodos para crear un DataSet:

- La forma más común es apuntar Spark a algunos archivos en sistemas de almacenamiento, usando la función **read** disponible en un SparkSession.
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people =
  spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- También puede crear un DataSet mediante la operación de transformation disponible en una existente. Por ejemplo, aplique la operación de map en un DataSet existente para crear un DataSet:
  ```
  val names = people.map(_.name) // In Scala: names is Dataset.
  Dataset<String> names = people.map((Person p) -> p.name,
  Encoders.STRING)); // Java
  ```

CLI y JDBCServer

Además de las API de programación, Spark SQL también proporciona las API de CLI/JDBC.

- Los scripts **spark-shell** y **spark-sql** pueden proporcionar la CLI para la depuración.
- JDBCServer proporciona API de JDBC. Los sistemas externos pueden enviar directamente solicitudes JDBC para calcular y analizar datos estructurados.

Principio de SparkSession

SparkSession es una API unificada en Spark2x y se puede considerar como una entrada unificada para la lectura de datos. SparkSession proporciona un único punto de entrada para

realizar muchas operaciones que anteriormente estaban dispersas entre varias clases, y también proporciona métodos de acceso a estas clases más antiguas para maximizar la compatibilidad.

Un SparkSession se puede crear usando un patrón de constructor. El constructor reutilizará automáticamente el SparkSession existente si hay un SparkSession o creará un SparkSession si no existe. Durante las transacciones de E/S, la configuración del elemento de configuración en el generador se sincroniza automáticamente con Spark y Hadoop.

```
import org.apache.spark.sql.SparkSession
val sparkSession = SparkSession.builder
  .master("local")
  .appName("my-spark-app")
  .config("spark.some.config.option", "config-value")
  .getOrCreate()
```

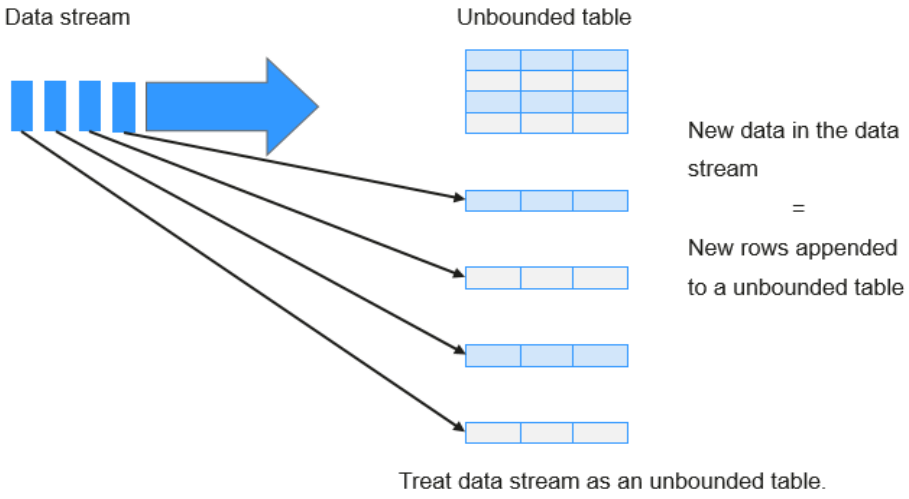
- SparkSession se puede usar para ejecutar consultas de SQL sobre datos y devolver resultados como DataFrame.
  ```
  sparkSession.sql("select * from person").show
  ```
- SparkSession se puede utilizar para establecer elementos de configuración durante la ejecución. Estos elementos de configuración se pueden reemplazar con variables en las sentencias de SQL.
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- SparkSession también incluye un método de "catalog" que contiene métodos para trabajar con Metastore (catálogo de datos). Después de utilizar este método, se devuelve un conjunto de datos, que se puede ejecutar utilizando la misma API de conjunto de datos.
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```
- Se puede acceder a SparkContext subyacente mediante la API de SparkContext de SparkSession.
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Principio de Structured Streaming

El Structured Streaming es un motor de procesamiento de flujo construido en el motor de Spark SQL. Puede usar la API Dataset/DataFrame en Scala, Java, Python o R para expresar agregaciones de streaming, ventanas de tiempo de eventos y stream-stream joins. Si los datos de streaming se producen de forma incremental y continua, Spark SQL continuará procesando los datos y sincronizando el resultado con el conjunto de resultados. Además, el sistema garantiza una tolerancia a fallas de extremo a extremo exactamente una vez a través de checkpoints y WALs.

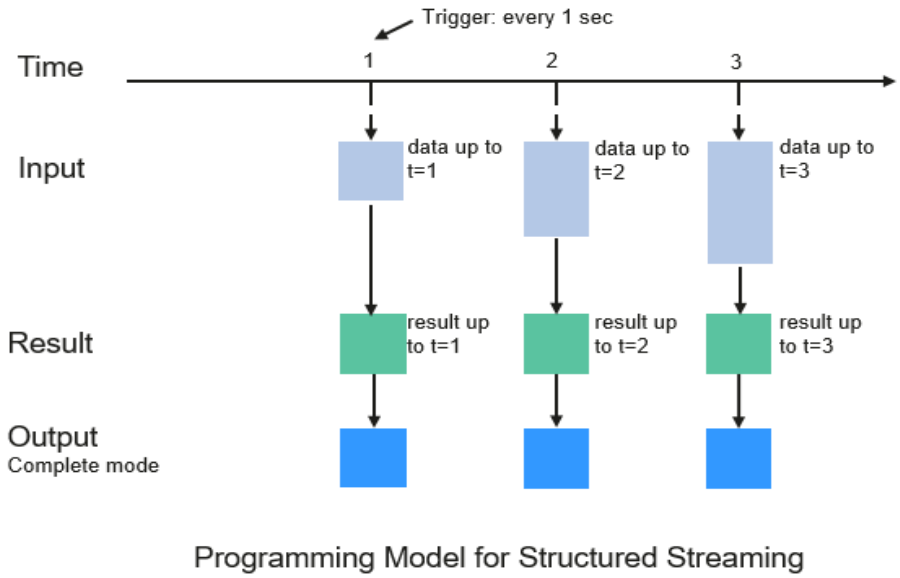
El núcleo del Structured Streaming es tomar los datos de streaming como una tabla de base de datos incremental. De manera similar al modelo de procesamiento de bloques de datos, el modelo de procesamiento de datos de flujo continuo aplica operaciones de consulta en una tabla de base de datos estática a la computación de flujo continuo, y Spark usa sentencias SQL estándar para consulta, para obtener datos de la tabla incremental y no delimitada.

Figura 6-109 Tabla no delimitada de Structured Streaming



Cada operación de consulta generará una tabla de resultados. En cada intervalo de activación, los datos actualizados se sincronizarán con la tabla de resultados. Cada vez que se actualiza la tabla de resultados, el resultado actualizado se escribirá en un sistema de almacenamiento externo.

Figura 6-110 Modelo de procesamiento de datos de Structured Streaming



Los modos de almacenamiento de Structured Streaming en la fase de output son los siguientes:

- Complete Mode: los conjuntos de resultados actualizados se escriben en el sistema de almacenamiento externo. La operación de escritura se realiza por un conector del sistema de almacenamiento externo.

- Append Mode: Si se activa un intervalo, solo los datos agregados en la tabla de resultados se escribirán en un sistema externo. Esto solo se aplica a las consultas en las que no se espera que cambien las filas existentes en la tabla de resultados.
- Update Mode: si se activa un intervalo, solo los datos actualizados en la tabla de resultados se escribirán en un sistema externo, que es la diferencia entre el Complete Mode y Update Mode.

Conceptos

- **RDD**

  El conjunto de datos distribuidos resilientes (RDD) es un concepto central de Spark. Indica un conjunto de datos distribuido de solo lectura y particionado. Los datos parciales o todos de este conjunto de datos pueden almacenarse en caché en la memoria y reutilizarse entre computaciones.

  Creación de RDD

  - Se puede crear un RDD a partir de la entrada de HDFS u otros sistemas de almacenamiento que sean compatibles con Hadoop.
  - Un nuevo RDD se puede convertir a partir de un RDD principal.
  - Un RDD se puede convertir a partir de una colección de conjuntos de datos a través de la codificación.

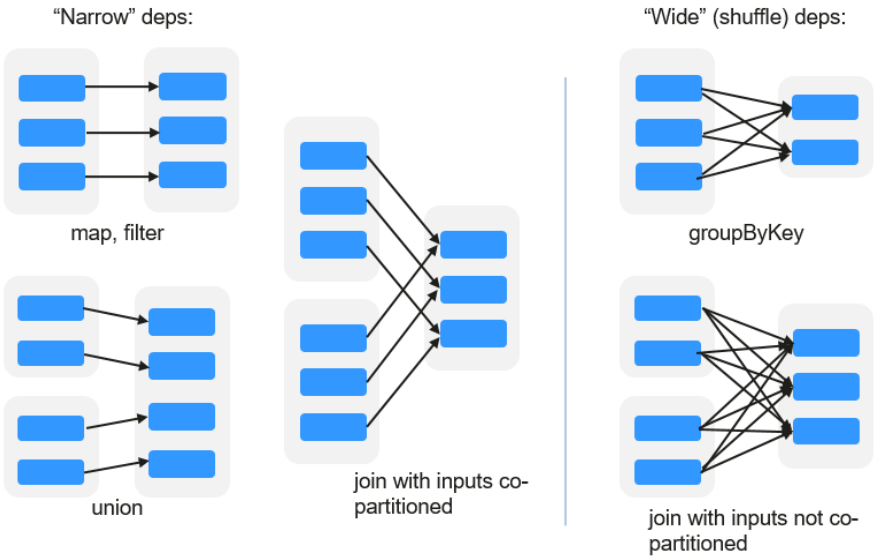
  Almacenamiento de RDD

  - Puede seleccionar diferentes niveles de almacenamiento para almacenar un RDD para su reutilización. (Hay 11 niveles de almacenamiento para almacenar un RDD.)
  - Por defecto, el RDD se almacena en la memoria. Cuando la memoria es insuficiente, el RDD se desborda al disco.

- **Dependencia de RDD**

  La dependencia de RDD incluye la dependencia estrecha y la dependencia amplia.

  Figura 6-111 Dependencia de RDD



  - **Dependencia estrecha**: Cada partición del RDD principal es utilizada como máximo por una partición del RDD secundario.

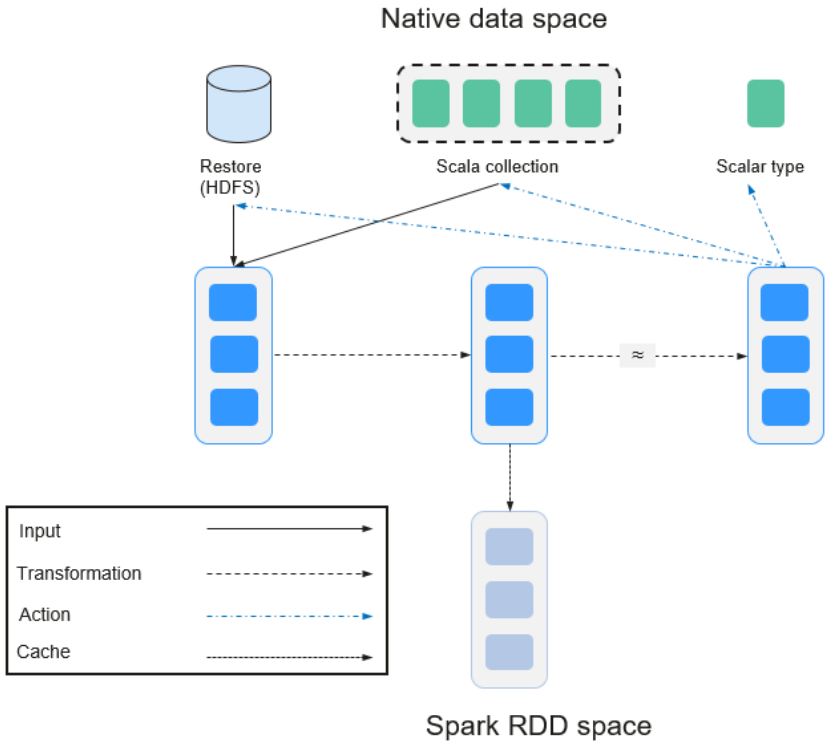
- **Dependencia amplia**: Las particiones del RDD secundaria dependen de todas las particiones del RDD principal.

La estrecha dependencia facilita la optimización. Lógicamente, cada operador de RDD es un fork/join (el join no es el operador de join mencionado anteriormente, sino la barrier utilizada para sincronizar múltiples tareas simultáneas); fork el RDD a cada partición y, a continuación, realiza el cálculo. Después de la computación, join los resultados y, a continuación, realice la operación de fork/join en el siguiente operador de RDD. Es antieconómico traducir directamente el RDD en implementación física. El primero es que cada RDD (incluso resultado intermedio) necesita ser físico en memoria o almacenamiento, lo que consume mucho tiempo y ocupa mucho espacio. El segundo es que, como barrier global, la operación de join es muy costosa y todo el proceso de join se ralentizará por el nodo más lento. Si las particiones del RDD secundaria dependen estrechamente de la del RDD principal, los dos procesos fork/join se pueden combinar para implementar la optimización de fusión clásica. Si la relación en la secuencia continua del operador es una dependencia estrecha, se pueden combinar múltiples procesos de fork/join para reducir un gran número de barriers globales y eliminar la fisicalización de muchos resultados intermedios de RDD, lo que mejora enormemente el rendimiento. Esto se llama optimización de canalización en Spark.

- **Transformation y Action (Operaciones de RDD)**

  Las operaciones en RDD incluyen transformation (el valor devuelto es un RDD) y action (el valor devuelto no es un RDD). **Figura 6-112** muestra el proceso de operación de RDD. El transformation es lazy, lo que indica que la transformación de un RDD a otro RDD no se ejecuta inmediatamente. Spark solo registra la transformation, pero no la ejecuta inmediatamente. La computación real se inicia solo cuando se inicia la acción. El action devuelve los resultados o escribe los datos RDD en el sistema de almacenamiento. Action es la fuerza motriz para que Spark inicie la computación.

Figura 6-112 Operación de RDD



Los datos y el modelo de operación de RDD son bastante diferentes de los de Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

a. El operador textFile lee los archivos de registro del HDFS y devuelve files (como un RDD).

b. El operador de filtro filtra las filas con **ERROR** y las asigna a errors (un nuevo RDD). El operador de filter es una transformation.

c. El operador de cache almacena en caché los errores para su uso futuro.

d. El operador de count devuelve el número de filas de errores. El operador de count es una acción.

Transformation incluye los siguientes tipos:

- Los elementos RDD se consideran elementos simples.

  La entrada y la salida tienen la relación uno a uno, y la estructura de partición del RDD resultante permanece sin cambios, por ejemplo, map.

  La entrada y la salida tienen la relación uno-a-muchos, y la estructura de partición del resultado RDD permanece sin cambios, por ejemplo, flatMap (un elemento se convierte en una secuencia que contiene varios elementos después del mapa y, a continuación, se aplana a varios elementos).

  La entrada y la salida tienen la relación uno a uno, pero la estructura de partición del resultado RDD cambia, por ejemplo, union (dos RDD se integran a un RDD, y el número de particiones se convierte en la suma del número de particiones de dos RDD) y coalescen (las particiones se reducen).

  Los operadores de algunos elementos se seleccionan de la entrada, tales como filter, distinct (los elementos duplicados se eliminan), subtract (los elementos solo existen en este RDD se conservan) y sample (se toman muestras).

- Los elementos RDD se consideran pares de clave-valor.

  Realizar el cálculo uno a uno en el único RDD, como mapValues (se conserva el modo de partición del RDD de origen, que es diferente del map).

  Ordenar el único RDD, como sort y partitionBy (partición con coherencia, lo que es importante para la optimización local).

  Reestructurar y reduce el RDD único basado en clave, como groupByKey y reduceByKey.

  Join y reestructurar dos RDD basados en la clave, como join y cogroup.

  > NOTA
  >
  > Las tres operaciones posteriores que implican la clasificación se denominan operaciones de shuffle.

Action incluye los siguientes tipos:

- Genere elementos de configuración escalar, como **count** (el número de elementos en el RDD devuelto), **reduce**, **fold/aggregate** (el número de elementos de configuración escalar que se devuelven) y **take** (el número de elementos antes del retorno).
- Generar la colección Scala, como **collect** (importar todos los elementos del RDD a la colección Scala) y **lookup** (buscar todos los valores corresponden a la clave).
- Escribir datos en el almacenamiento, como **saveAsTextFile** (que corresponde al **textFile** anterior).

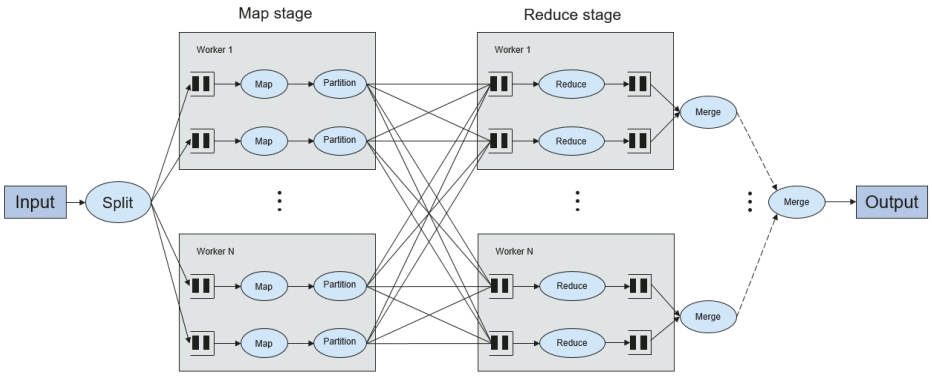
- Puntos de comprobación, como el operador **checkpoint**. Cuando el lineage es bastante largo (lo que ocurre con frecuencia en el cálculo de gráficos), se necesita un largo período de tiempo para ejecutar toda la secuencia de nuevo cuando se produce un falla. En este caso, checkpoint se utiliza como el punto de control para escribir los datos actuales en el almacenamiento estable.

- **Shuffle**

  La mezcla aleatoria es una fase específica en el marco de MapReduce, que se encuentra entre la fase Map y la fase Reduce. Si los resultados de salida de Map van a ser utilizados por Reduce, los resultados de salida deben ser hashed basados en una clave y distribuidos a cada Reducer. Este proceso se llama Shuffle. Shuffle implica la lectura y escritura del disco y la transmisión de la red, de modo que el rendimiento de Shuffle afecta directamente a la eficiencia de operación de todo el programa.

  La siguiente figura muestra todo el proceso del algoritmo de MapReduce.

  Figura 6-113 Proceso de algoritmo

  Shuffle es un puente para conectar datos. A continuación se describe la implementación de shuffle en Spark.

  Shuffle divide un job de Spark en múltiples stages. Las primeras etapas contienen uno o más ShuffleMapTasks y la última stage contiene uno o más ResultTasks.

- **Estructura de Spark Application**

  La estructura de aplicación Spark incluye el SparkContext inicializado y el programa principal.

  - SparkContext inicializado: construye el entorno operativo de la aplicación Spark.

    Construye el objeto de SparkContext. A continuación se presenta un ejemplo:

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    Descripción de parámetros:

    master indica la cadena de enlace. Los modos de enlace incluyen local, Yarn-cluster y Yarn-cliente.

    appName indica el nombre de la aplicación.

    SparkHome indica el directorio donde está instalado Spark en el clúster.

    jars indica el código y el paquete de dependencias de una aplicación.

  - Programa principal: procesa datos.

  Para obtener más información sobre cómo presentar una solicitud, visite **https://spark.apache.org/docs/3.1.1/submitting-applications.html**.

- **Comandos de Spark Shell**

  Los comandos básicos de Spark shell soportan el envío de aplicaciones de Spark. Los comandos de shell Spark son los siguientes:

  ```
  ./bin/spark-submit \
    --class <main-class> \
    --master <master-url> \
    ... # other options
    <application-jar> \
    [application-arguments]
  ```

  Descripción de parámetros:

  --class: indica el nombre de la clase de un Spark application.

  --master: indica el patrón al que se vincula la aplicación de Spark, como Yarn-client y Yarn-cluster.

  application-jar: indica la ruta del archivo JAR de Spark application.

  application-arguments: indica el parámetro necesario para enviar Spark application. Este parámetro se puede dejar en blanco.

- **Spark JobHistory Server**

  La interfaz de usuario web de Spark se utiliza para monitorear los detalles en cada fase de marco de Spark de un trabajo de Spark en ejecución o histórico y proporcionar la visualización del registro, lo que ayuda a los usuarios a desarrollar, configurar y optimizar el trabajo en unidades más finas.

6.28.2 Solución de Spark2x HA

6.28.2.1 Instancia multiactiva de Spark2x

Fondo

Basado en JDBCServers existentes en la comunidad, HA de instancia multiactiva se utiliza para lograr la alta disponibilidad. En este modo, varios JDBCServers coexisten en el clúster y el cliente puede conectar aleatoriamente cualquier JDBCServer para realizar operaciones de servicio. Cuando uno o varios JDBCServers dejan de funcionar, un cliente puede conectarse a otro JDBCServer normal.

En comparación con el HA activo/en espera, el HA de instancia multiactivo elimina las siguientes restricciones:

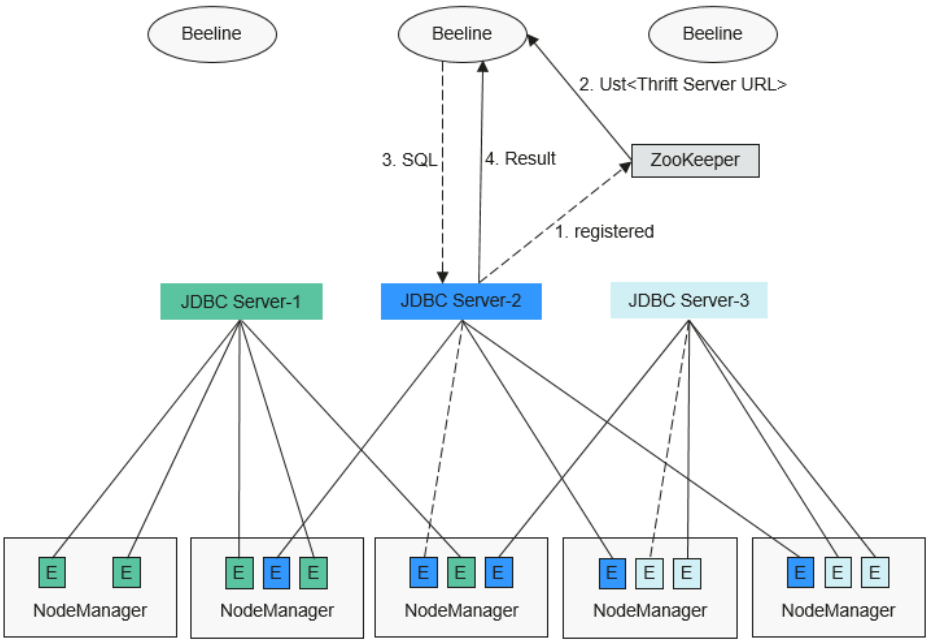
- En HA activo/en espera, cuando se produce la conmutación activa/en espera, el período no disponible no puede ser controlado por JDBCServer, sino determinado por los recursos de servicio de Yarn.
- En Spark, el Thrift JDBC similar a HiveServer2 proporciona servicios y los usuarios acceden a los servicios a través de Beeline y JDBC API. Por lo tanto, la capacidad de procesamiento del clúster de JDBCServer depende de la capacidad de punto único del servidor primario, y la escalabilidad es insuficiente.

El HA de instancia multiactiva no solo evita la interrupción del servicio causada por la conmutación, sino que también permite el escalamiento horizontal del clúster para asegurar una alta concurrencia.

Implementación

La siguiente figura muestra el principio básico de HA de instancia multiactiva de Spark JDBCServer.

Figura 6-114 Spark JDBCServer HA



1. Después de iniciar JDBCServer, se registra con ZooKeeper escribiendo información de nodo en un directorio especificado. La información del nodo incluye la IP de la instancia de JDBCServer, el número de puerto, la versión y el número de serie (la información de los diferentes nodos está separada por comas).

   Un ejemplo es el siguiente:

   ```
   [serverUri=192.168.169.84:22550
   ;version=8.1.0.1;sequence=0000001244,serverUri=192.168.195.232:22550 ;version=
   8.1.0.1;sequence=0000001242,serverUri=192.168.81.37:22550 ;version=8.1.0.1;seq
   uence=0000001243]
   ```

2. Para conectarse a JDBCServer, el cliente debe especificar el espacio de nombres, que es el directorio de las instancias de JDBCServer en ZooKeeper. Durante la conexión, se selecciona aleatoriamente una instancia de JDBCServer del espacio de nombres especificado. Para obtener más información sobre la URL, consulte **Conexión de URL**.
3. Una vez que la conexión se realiza correctamente, el cliente envía sentencias de SQL a JDBCServer.
4. JDBCServer ejecuta sentencias de SQL recibidas y envía los resultados al cliente.

En el modo HA de instancia multiactiva, todas las instancias de JDBCServer son independientes y equivalentes. Cuando una instancia se interrumpe durante la actualización, otras instancias de JDBCServer pueden aceptar la solicitud de conexión del cliente.

Se deben seguir las siguientes reglas en la instancia multiactiva HA de Spark JDBCServer:

- Si una instancia de JDBCServer sale de forma anormal, ninguna otra instancia se hará cargo de las sesiones y servicios que se ejecutan en esta instancia anormal.
- Cuando se detiene el proceso de JDBCServer, los nodos correspondientes se eliminan de ZooKeeper.

- El cliente selecciona aleatoriamente el servidor, lo que puede dar como resultado una asignación de sesión desigual, y finalmente dar como resultado un desequilibrio de carga de instancia.
- Después de que la instancia entra en el modo de mantenimiento (en el que no se acepta ninguna nueva solicitud de conexión del cliente), los servicios que aún se ejecutan en la instancia pueden fallar cuando se agota el tiempo de desmantelamiento.

Conexión de URL

Modo de instancia multiactiva

En el modo de instancia multiactiva, el cliente lee el contenido del nodo ZooKeeper y se conecta a JDBCServer. Las cadenas de conexión son las siguientes:

- Modo de seguridad:
  - Si la autenticación de Kinit está habilitada, JDBCURL es la siguiente:

    ```
    jdbc:hive2://
    <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<System domain name>@<System domain name>;
    ```

    NOTA

    - **<zkNode_IP>:<zkNode_Port>** indica la URL del ZooKeeper. Utilice comas (,) para separar varias URL,

      Por ejemplo, **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
    - **sparkthriftserver2x** indica el directorio de ZooKeeper donde una instancia JDBCServer aleatoria está conectada al cliente.

    Por ejemplo, cuando utiliza el cliente Beeline para la conexión en modo de seguridad, ejecute el siguiente comando:

    sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.**<*System domain name*>**@**<*System domain name*>**;"**
  - Si la autenticación de Keytab está habilitada, JDBCURL es la siguiente:

    ```
    jdbc:hive2://
    <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<System domain name>@<System domain name>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
    ```

    <*principal_name*> indica el principal del usuario de Kerberos, por ejemplo, **test@**<*System domain name*>. <*path_to_keytab*> indica la ruta del archivo Keytab correspondiente a <*principal_name*> como, por ejemplo, **/opt/auth/test/user.keytab**.
- Modo común:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
  ```

  Por ejemplo, cuando utiliza el cliente de Beeline para la conexión en modo común, ejecute el siguiente comando:

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*z*

kNode3_Port>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;"**

Modo de instancia no multiactiva

En el modo de instancia no multiactiva, un cliente se conecta a un nodo de JDBCServer especificado. En comparación con el modo de instancia multiactiva, la cadena de conexión en el modo de instancia no multiactiva no contiene los parámetros **serviceDiscoveryMode** y **zooKeeperNamespace** sobre ZooKeeper.

Por ejemplo, cuando utiliza el cliente de Beeline para conectar JDBCServer en modo de instancia no multiactivo, ejecute el siguiente comando:

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*server_IP*>:<*server_Port*>/**;user.principal=spark2x/hadoop.**<*System domain name*>@<*System domain name*>**;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.**<*System domain name*>@<*System domain name*>**;"**

NOTA

- **<server_IP>:<server_Port>** indica la dirección URL del nodo de JDBCServer especificado.
- **CLIENT_HOME** indica la ruta del cliente.

Excepto el método de conexión, las operaciones de la API de JDBCServer en modo de instancia multiactiva y modo de instancia no multiactiva son las mismas. Spark JDBCServer es otra implementación de HiveServer2 en Hive. Para obtener más información sobre cómo usar Spark JDBCServer, visite el sitio web oficial de Hive en **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

6.28.2.2 Multitenant de Spark2x

Fondo

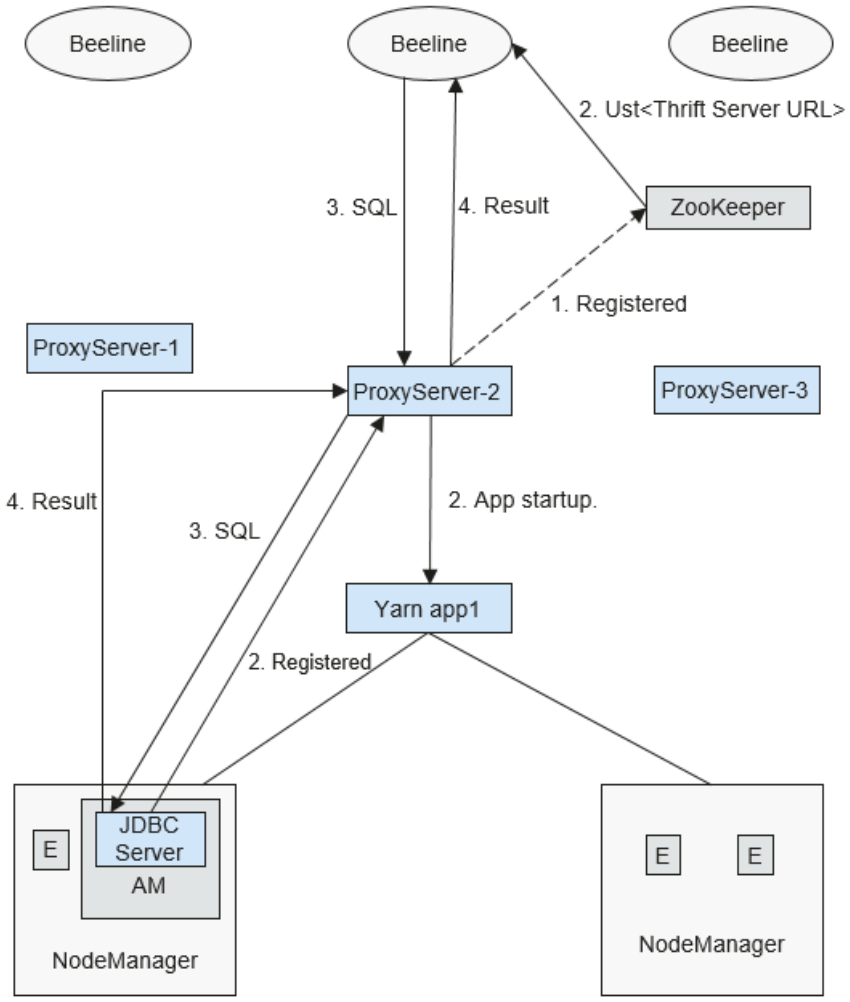
En el modo de instancia multiactiva de JDBCServer, JDBCServer implementa el modo de Yarn-client, pero solo hay disponible una cola de recursos de Yarn. Para resolver el problema de limitación de recursos, se introduce el modo multitetant.

En el modo multitenant, JDBCServers están vinculados con tenants. Cada tenant corresponde a uno o más JDBCServers, y un JDBCServer proporciona servicios para un solo tenant. Se pueden configurar diferentes tenants con diferentes colas de Yarn para implementar el aislamiento de recursos. Además, JDBCServer puede iniciarse dinámicamente según sea necesario para evitar el desperdicio de recursos.

Implementación

Figura 6-115 muestra la solución de HA del modo multitenant.

Figura 6-115 Modo multitenant de Spark JDBCServer



1. Cuando se inicia ProxyServer, se registra con ZooKeeper escribiendo información de nodo en un directorio especificado. La información de nodo incluye la IP de instancia, el número de puerto, la versión y el número de serie (la información de diferentes nodos está separada por comas).

   NOTA

   En el modo multitenant, la instancia de JDBCServer en la página MRS indica ProxyServer, el agente de JDBCServer.

   Un ejemplo es el siguiente:

   ```
   serverUri=192.168.169.84:22550
   ;version=8.1.0.1;sequence=0000001244,serverUri=192.168.195.232:22550
   ;version=8.1.0.1;sequence=0000001242,serverUri=192.168.81.37:22550
   ;version=8.1.0.1;sequence=0000001243,
   ```

2. Para conectarse a ProxyServer, el cliente debe especificar un espacio de nombres, que es el directorio de la instancia de ProxyServer a la que desea acceder en ZooKeeper. Cuando el cliente se conecta a ProxyServer se selecciona aleatoriamente una instancia en Namespace para la conexión. Para obtener más información sobre la dirección URL, consulte **Conexión de URL**.
3. Una vez que el cliente se conecta correctamente a ProxyServer, ProxyServer comprueba si existe el JDBCServer de un tenant. En caso afirmativo, Beeline conecta el

JDBCServer. Si no, se inicia un nuevo JDBCServer en modo de Yarn-cluster. Después del inicio de JDBCServer, ProxyServer obtiene la dirección IP del JDBCServer y establece la conexión entre Beeline y JDBCServer.

4. El cliente envía sentencias de SQL a ProxyServer que luego reenvía sentencias al JDBCServer conectado. JDBCServer devuelve los resultados a ProxyServer que luego devuelve los resultados al cliente.

En el modo multitenant de HA, todas las instancias de ProxyServer son independientes y equivalentes. Si una instancia se interrumpe durante la actualización, otras instancias pueden aceptar la solicitud de conexión del cliente.

Conexión de URL

Modo multitenant

En modo multitenant, el cliente lee el contenido del nodo de ZooKeeper y se conecta a ProxyServer. Las cadenas de conexión son las siguientes:

- Modo de seguridad:
  - Si la autenticación de Kinit está habilitada, la URL del cliente es la siguiente:

    ```
    jdbc:hive2://
    <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<System domain name>@<System domain name>;
    ```

    NOTA

    - **<zkNode_IP>:<zkNode_Port>** indica la URL del ZooKeeper. Utilice comas (,) para separar varias URL,

      Por ejemplo, **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
    - **sparkthriftserver2x** indica el directorio de ZooKeeper donde una instancia de JDBCServer aleatoria está conectada al cliente.

    Por ejemplo, cuando utiliza el cliente Beeline para la conexión en modo de seguridad, ejecute el siguiente comando:

    sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.**<*System domain name*>**@**<*System domain name*>**;"**
  - Si la autenticación de Keytab está habilitada, la URL es la siguiente:

    ```
    jdbc:hive2://
    <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<System domain name>@<System domain name>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
    ```

    <*principal_name*> indica el principal del usuario de Kerberos, por ejemplo, **test@**<*System domain name*>. <*path_to_keytab*> indica la ruta del archivo Keytab correspondiente a <*principal_name*> como, por ejemplo, **/opt/auth/test/user.keytab**.
- Modo común:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
  ```


Por ejemplo, cuando utiliza el cliente de Beeline para la conexión en modo común, ejecute el siguiente comando:

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*zkNode1_IP*>:<*zkNode1_Port*>,<*zkNode2_IP*>:<*zkNode2_Port*>,<*zkNode3_IP*>:<*zkNode3_Port*>/**;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;"**

Modo no multitenant

En modo no multitenant, un cliente se conecta a un nodo de JDBCServer especificado. En comparación con el modo de instancia multiactiva, la cadena de conexión en el modo de instancia no multiactiva no contiene los parámetros **serviceDiscoveryMode** y **zooKeeperNamespace** sobre ZooKeeper.

Por ejemplo, cuando utiliza el cliente de Beeline para conectar JDBCServer en modo de instancia de non-multitenant, ejecute el siguiente comando:

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*server_IP*>:<*server_Port*>/**;user.principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.**<*System domain name*>@<*System domain name*>**;"**

NOTA

- **<server_IP>:<server_Port>** indica la dirección URL del nodo de JDBCServer especificado.
- **CLIENT_HOME** indica la ruta del cliente.

Excepto el método de conexión, otras operaciones de JDBCServer API en modo de multitenant y modo non-multitenant son las mismas. Spark JDBCServer es otra implementación de HiveServer2 en Hive. Para obtener más información sobre cómo usar Spark JDBCServer, visite el sitio web oficial de Hive en **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

Especificación de un tenant

Generalmente, el cliente enviado por un usuario se conecta al JDBCServer predeterminado del tenant al que pertenece el usuario. Si desea conectar el cliente al JDBCServer de un tenant especificado, agregue el parámetro **--hiveconf mapreduce.job.queuename**.

El comando para conectar Beeline es el siguiente (**aaa** indica el nombre del tenant):

beeline --hiveconf mapreduce.job.queuename=aaa -u 'jdbc:hive2://192.168.39.30:2181,192.168.40.210:2181,192.168.215.97:2181;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<*System domain name*>@<*System domain name*>;'

6.28.3 Relación entre Spark2x y otros componentes

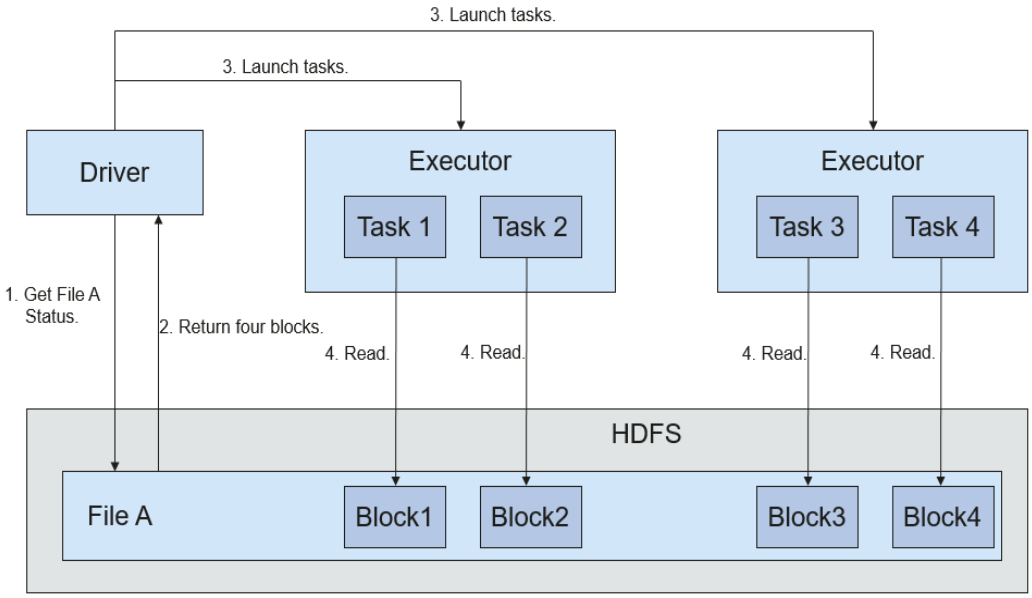
Relación entre Spark y HDFS

Los datos calculados por Spark provienen de múltiples fuentes de datos, como archivos locales y HDFS. La mayoría de los datos provienen de HDFS que pueden leer datos a gran escala para computación paralela Después de ser computados, los datos se pueden almacenar en HDFS.

Spark implica Driver y Executor. El Driver programa las tareas y el Executor ejecuta las tareas.

Figura 6-116 describe el proceso de lectura de archivos.

Figura 6-116 Proceso de lectura de archivos

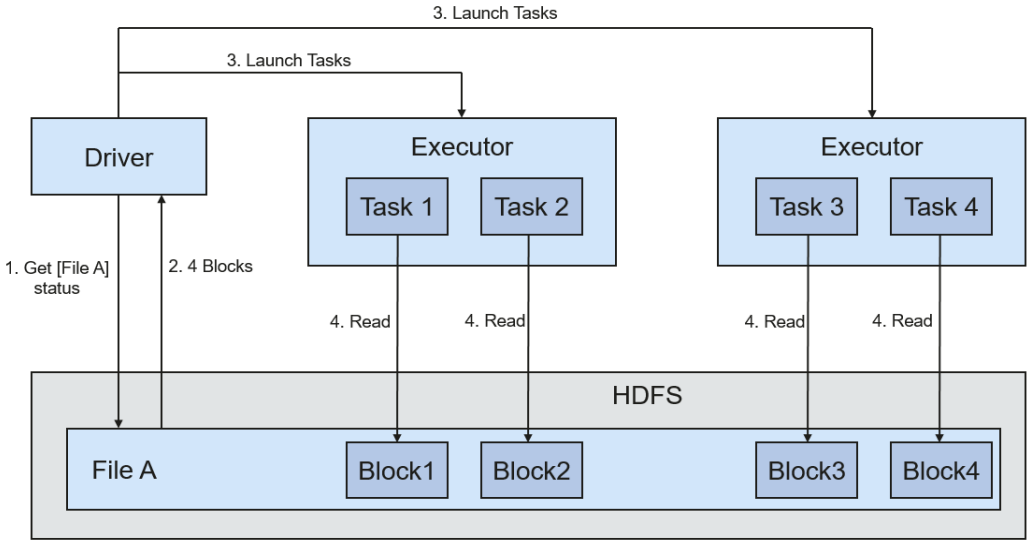


El proceso de lectura de archivos es el siguiente:

1. Driver se interconecta con HDFS para obtener la información del File A.
2. El HDFS devuelve la información de block detallada acerca de este archivo.
3. Driver establece un grado paralelo basado en la cantidad de datos de block y crea varias tasks para leer los blocks de este archivo.
4. Executor ejecuta las tareas y lee los blocks detallados como parte del conjunto de datos distribuido resistentes (RDD).

Figura 6-117 describe el proceso de escritura de archivos.

Figura 6-117 Proceso de escritura de archivos



El proceso de escritura de archivos es el siguiente:

1. Driver crea un directorio donde se va a escribir el archivo.
2. Basándose en el estado de distribución de RDD, se calcula el número de tasks relacionadas con la escritura de datos, y estas tareas se envían al Executor.
3. Executor ejecuta estas tareas y escribe los datos de RDD en el directorio creado en **1**.

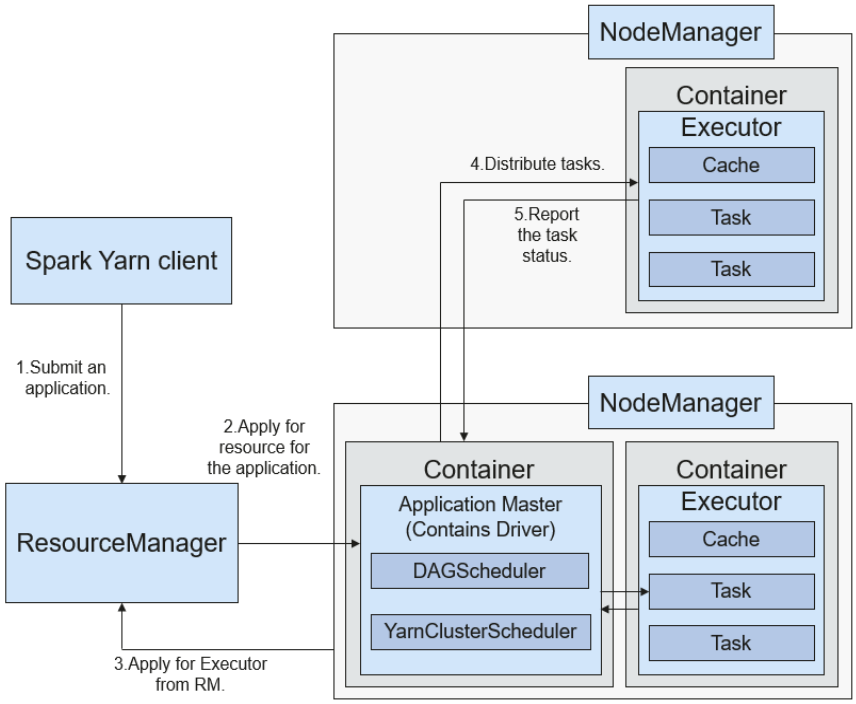
Relación con Yarn

La computación y programación de Spark pueden implementarse usando el modo de Yarn. Spark disfruta de los recursos informáticos proporcionados por los clústeres de Yarn y ejecuta tareas de forma distribuida. Spark en Yarn tiene dos modos: Yarn-cluster y Yarn-client.

- Modo de Yarn-cluster

  Figura 6-118 describe el marco de operación.

Figura 6-118 Marco de operación de Spark on Yarn-cluster



Proceso de implementación de Spark on Yarn-cluster:

a. El cliente genera la información de la aplicación y, a continuación, envía la información a ResourceManager.

b. ResourceManager asigna el primer container (ApplicationMaster) a SparkApplication e inicia el driver en el container.

c. ApplicationMaster se aplica a los recursos de ResourceManager para ejecutar el container.

   ResourceManager asigna los containers a ApplicationMaster, que se comunica con los NodeManagers relacionados e inicia el executor en el container obtenido. Después de iniciar el executor, se registra con drivers y se aplica a las tareas.

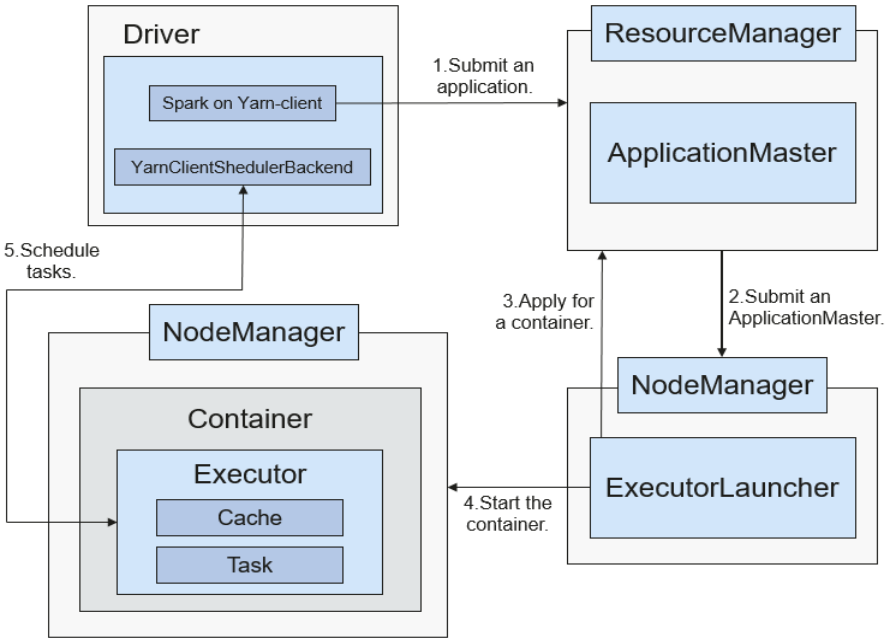
d. Los controladores asignan tareas a los ejecutores.

e. Los executors ejecutan tareas e informan el estado de funcionamiento a Drivers.

- Modo de Yarn-client

  Figura 6-119 describe el marco de operación.

Figura 6-119 Marco de operación de Spark on Yarn-client



Proceso de implementación de Spark on Yarn-client:

NOTA

En el modo Yarn-cliente, el Driver se despliega y se inicia en el client. En el modo Yarn-client, el cliente de una versión anterior es incompatible. Se recomienda el modo Yarn-cluster.

a. El cliente envía la solicitud de aplicación Spark a ResourceManager y empaqueta toda la información necesaria para iniciar ApplicationMaster y envía la información a ResourceManager. A continuación, ResourceManager devuelve los resultados al cliente. Los resultados incluyen información como ApplicationId y el límite superior, así como el límite inferior de los recursos disponibles. Después de recibir la solicitud, ResourceManager encuentra un nodo apropiado para ApplicationMaster y lo inicia en este nodo. ApplicationMaster es un papel en Yarn, y el nombre del proceso en Spark es ExecutorLauncher.

b. En función de los requerimientos de recursos de cada tarea, ApplicationMaster puede solicitar una serie de containers para ejecutar tareas desde ResourceManager.

c. Después de recibir la lista de container recién asignados (de ResourceManager), ApplicationMaster envía información a los NodeManagers relacionados para iniciar los containers.

   ResourceManager asigna los containers a ApplicationMaster, que se comunica con los NodeManagers relacionados e inicia el executor en el container obtenido. Después de iniciar el executor, se registra con drivers y se aplica a las tareas.

   NOTA

   Containers en ejecución no se suspenderá para liberar recursos.

d. Drivers asignan task a los executors. Los executors ejecutan tareas e informan el estado de funcionamiento a Drivers.

6.28.4 Nuevas funciones de código abierto de Spark2x

Objetivo

En comparación con Spark 1.5, Spark2*x* tiene algunas nuevas características de código abierto. Las características o conceptos específicos son los siguientes:

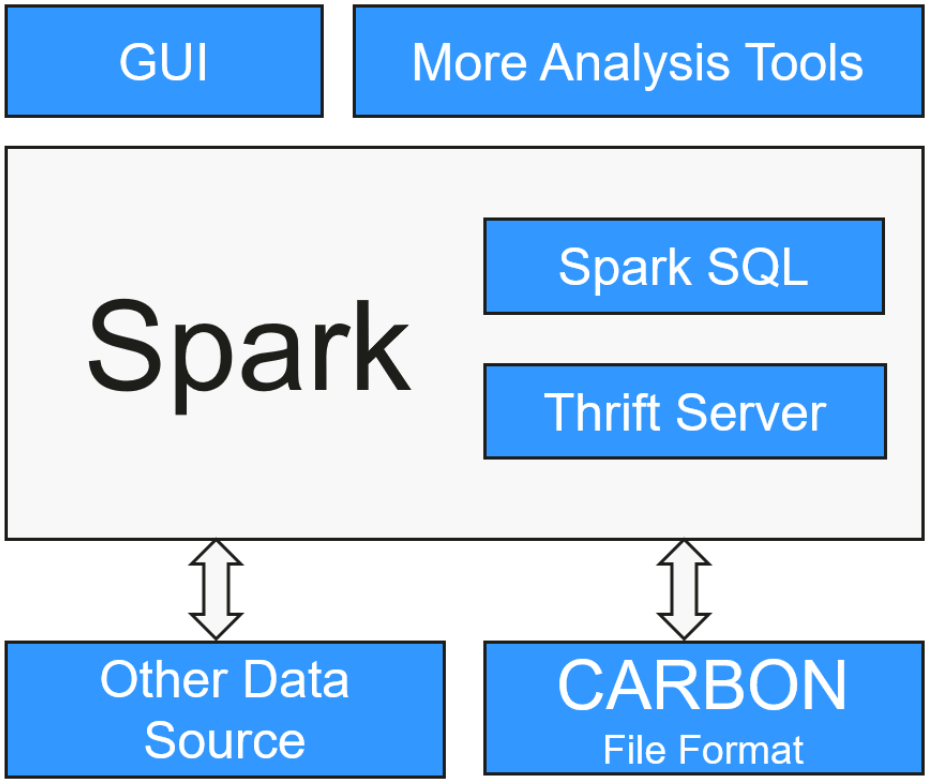
- DataSet: Para más información, consulte **Principio de SparkSQL y DataSet**.
- Spark SQL Native DDL/DML: Para obtener más información, consulte **Principio de SparkSQL y DataSet**.
- SparkSession: Para más información, consulte **Principio de SparkSession**,
- Structured Streaming: Para obtener más información, consulte **Principio de Structured Streaming**.
- Optimización de archivos pequeños
- Optimización del algoritmo agregado
- Optimización de tabla de Datasource
- Fusión de CBO

6.28.5 Funciones de código abierto mejoradas de Spark2x

6.28.5.1 Descripción de CarbonData

CarbonData es un nuevo formato de almacén de datos nativo de Apache Hadoop. CarbonData permite consultas interactivas más rápidas sobre PetaBytes de datos utilizando técnicas avanzadas de almacenamiento de columnas, índice, compresión y codificación para mejorar la eficiencia informática. Además, el CarbonData es también un motor de análisis de alto rendimiento que integra fuentes de datos con Spark.

Figura 6-120 Arquitectura básica de CarbonData



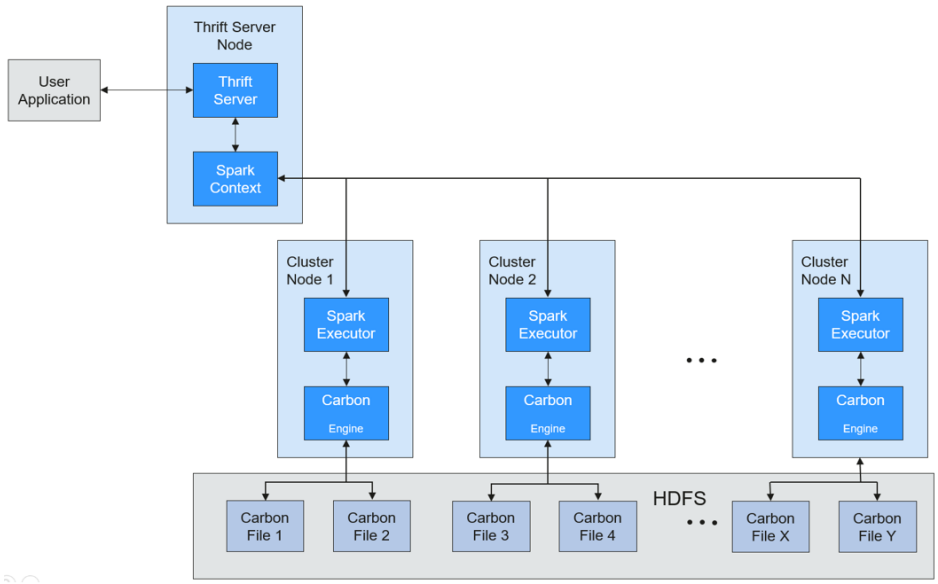
El propósito de utilizar CarbonData es proporcionar una respuesta rápida a consultas de ad hoc de big data. CarbonData es un motor de procesamiento analítico en línea (OLAP), que almacena datos utilizando tablas similares a las del Sistema de gestión de bases de datos relacionales (RDBMS). Puede importar más de 10 TB de datos a tablas creadas en formato CarbonData y CarbonData organiza y almacena automáticamente los datos mediante los índices multidimensionales comprimidos. Después de cargar los datos en CarbonData, CarbonData responde a consultas de ad hoc en segundos.

CarbonData integra fuentes de datos en el ecosistema de Spark y puede consultar y analizar los datos con Spark SQL. También puede utilizar la herramienta de terceros JDBCServer proporcionada por Spark para conectarse a SparkSQL.

Topología de CarbonData

CarbonData se ejecuta como una fuente de datos dentro de Spark. Por lo tanto, CarbonData no inicia ningún proceso adicional en los nodos de los clústeres. CarbonData engine funciona dentro del Spark executor.

Figura 6-121 Topología de CarbonData



Los datos almacenados en CarbonData Table se dividen en varios archivos de datos de CarbonData. Cada vez que se consultan datos, CarbonData Engine lee y filtra conjuntos de datos. CarbonData Engine se ejecuta como parte del proceso Spark Executor y es responsable de manejar un subconjunto de bloques de archivos de datos.

Los datos de la tabla se almacenan en HDFS. Los nodos del mismo clúster de Spark se pueden utilizar como nodos de datos de HDFS.

Características de CarbonData

- SQL CarbonData es compatible con Spark SQL y admite operaciones de consulta SQL realizadas en Spark SQL.
- Definición de conjunto de datos de table simple: CarbonData le permite definir y crear conjuntos de datos mediante sentencias de lenguaje de definición de datos (DDL) fáciles de usar. CarbonData DDL es flexible y fácil de usar, y puede definir tables complejos.
- Fácil gestión de datos: CarbonData ofrece varias funciones de gestión de datos para la carga y el mantenimiento de datos. CarbonData admite la carga masiva de datos históricos y la carga incremental de nuevos datos. Los datos cargados se pueden eliminar en función del tiempo de carga y se puede deshacer una operación de carga específica.
- El formato de archivo CarbonData es un almacén de columnas en HDFS. Este formato tiene muchas nuevas características de almacenamiento de archivos basadas en columnas, como la división de tablas y la compresión de datos. CarbonData tiene las siguientes características:
  - Almacena datos junto con el índice: Acelera significativamente el rendimiento de la consulta y reduce los análisis de E/S y los recursos de CPU, cuando hay filtros en la consulta. El índice de CarbonData consta de múltiples niveles de índices. Un marco de procesamiento puede aprovechar este índice para reducir la tarea que necesita programar y procesar, y también puede realizar omitir el escaneo en una unidad de grano más fino (llamada blocklet) en el escaneo del lado de la tarea en lugar de escanear todo el archivo.
  - Datos codificados operables: A través de la compatibilidad con esquemas de compresión y codificación globales eficientes, CarbonData puede consultar datos

comprimidos/codificados. Los datos se pueden convertir justo antes de devolver los resultados a los usuarios, lo que se denomina materialización tardía.

- Soporta varios casos de uso con un solo formato de datos: como consulta interactiva de estilo OLAP, acceso secuencial (big scan) y acceso aleatorio (narrow scan).

Tecnologías clave y ventajas de CarbonData

- Respuesta rápida a la consulta: CarbonData ofrece consultas de alto rendimiento. La velocidad de consulta de CarbonData es 10 veces mayor que la de Spark SQL. Utiliza formatos de datos dedicados y aplica múltiples tecnologías de índice, código de diccionario global y múltiples optimizaciones push-down, proporcionando una respuesta rápida a las consultas de datos a nivel de TB.
- Compresión de datos eficiente: CarbonData comprime los datos combinando los algoritmos de compresión ligeros y pesados. Esto ahorra significativamente entre un 60% y un 80% de espacio de almacenamiento de datos y el costo de almacenamiento de hardware.

Servidor de caché de índices de CarbonData

Para resolver la presión y los problemas provocados por el aumento del volumen de datos al driver, se introduce un servidor de caché de índice independiente para separar el índice del lado de aplicación Spark de la consulta Carbon. Todo el contenido del índice es gestionado por el servidor de caché de índices. Las aplicaciones de Spark obtienen los datos de índice requeridos en modo RPC. De esta manera, se libera una gran cantidad de memoria en el lado del servicio para que los servicios no se vean afectados por la escala del clúster y el rendimiento o las funciones no se vean afectados.

6.28.5.2 Optimización de la consulta SQL de datos de múltiples fuentes

Escenario

Las empresas suelen almacenar datos masivos, como de varias bases de datos y almacenes, para la gestión y la recopilación de información. Sin embargo, las fuentes de datos diversificadas, las estructuras de conjuntos de datos híbridos y el almacenamiento de datos dispersos reducen la eficiencia de las consultas.

El Spark de código abierto solo admite un simple pushdown de filtro durante la consulta de datos de múltiples fuentes. El rendimiento del motor SQL se deteriora debido a una gran cantidad de transmisión de datos innecesaria. La función pushdown se mejora, de modo que **aggregate**, **projection** complejo, y**predicate** complejo se pueden enviar a las fuentes de datos, reduciendo la transmisión de datos innecesaria y mejorando el rendimiento de la consulta.

Solo el origen de datos JDBC admite operaciones de consulta, como **aggregate**, **projection**, **predicate**, **aggregate over inner join** y **aggregate over union all**. Todas las operaciones de pushdown se pueden habilitar en función de sus necesidades.

Tabla 6-25 Consulta mejorada de consulta entre fuentes

Módulo	Antes de la mejora	Después de la mejora
aggregate	El pushdown de **aggregate** no es compatible.	• Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. Ejemplo: select count(*) from table • Se admiten expresiones internas de funciones de agregación. Ejemplo: select sum(a+b) from table • Se admite el cálculo de funciones de agregación. Ejemplo: select avg(a) + max(b) from table • Se admite pushdown de **having**. Ejemplo: select sum(a) from table where a>0 group by b having sum(a)>10 • Se soporta el pushdown de algunas funciones. Se admite la reducción de líneas en matemáticas, tiempo y funciones de cadena, como **abs()**, **month()** y **length()**. Además de las funciones integradas anteriores, puede ejecutar el comando **SET** para agregar funciones compatibles con los orígenes de datos. Ejemplo: select sum(abs(a)) from table • Se admite pushdown de **limit** y **order by** después de **aggregate**. Sin embargo, Oracle no admite el pushdown, ya que Oracle no admite **limit**. Ejemplo: select sum(a) from table where a>0 group by b order by sum(a) limit 5
projection	Solo se soporta el pushdown de **projection** simple. Ejemplo: select a, b from table	• Las expresiones complejas pueden ser presionadas hacia abajo. Ejemplo: select (a+b)*c from table • Algunas funciones pueden ser presionadas hacia abajo. Para obtener más información, consulte la descripción que se encuentra debajo de la tabla. Ejemplo: select length(a)+abs(b) from table • Se admite pushdown de **limit** y **order by** después de **projection**. Ejemplo: select a, b+c from table order by a limit 3

Módulo	Antes de la mejora	Después de la mejora
predicate	Solo se admite el filtrado simple con el nombre de columna a la izquierda del operador y los valores a la derecha. Ejemplo: seleccione * de la tabla donde a>0 o b en ("aaa", "bbb")	● Se admite el pushdown de expresiones complejas. Ejemplo: select * from table where a +b>c*d or a/c in (1, 2, 3) ● Algunas funciones pueden ser presionadas hacia abajo. Para obtener más información, consulte la descripción que se encuentra debajo de la tabla. Ejemplo: select * from table where length(a)>5
aggregate over inner join	Los datos relacionados de las dos tablas deben cargarse en Spark. La operación de join debe realizarse antes de la operación **aggregate**.	Se soportan las siguientes funciones: ● Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. ● Todas las operaciones de **aggregate** se pueden realizar en una misma tabla. Las operaciones de **group by** se pueden realizar en una o dos tablas y solo se admite el inner join. No se admiten los siguientes escenarios: ● **aggregate** no se puede empujar hacia abajo desde las tablas de join izquierda y derecha. ● **aggregate** contiene operaciones, por ejemplo, sum (a+b). ● Operaciones de **aggregate**, por ejemplo, sum(a)+min(b).
aggregate over union all	Los datos relacionados de las dos tablas deben cargarse en Spark. **union** debe realizarse antes de **aggregate**.	Escenarios admitidos: Funciones de agregación que incluyen **sum**, **avg**, **max**, **min** y **count** son compatibles. Escenarios no admitidos: ● **aggregate** contiene operaciones, por ejemplo, sum (a+b). ● Operaciones de **aggregate**, por ejemplo, sum(a)+min(b).

Precauciones

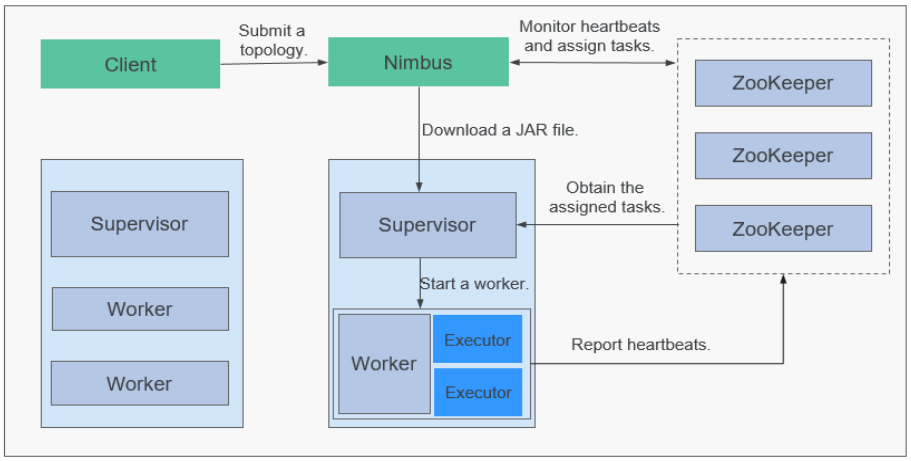
- Si el origen de datos externo es Hive, la operación de consulta no se puede realizar en tablas externas creadas por Spark.
- Solo se admiten las fuentes de datos MySQL y MPPDB.

6.29 Storm

6.29.1 Principios básicos de Storm

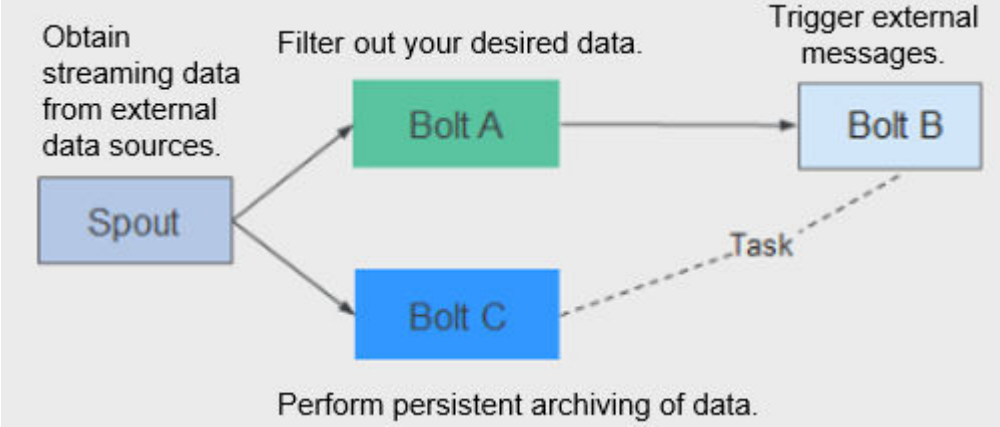
Apache Storm es un sistema de procesamiento de datos en tiempo real distribuido, confiable y tolerante a fallas. En Storm, una estructura de datos en forma de gráfico llamada topología necesita ser diseñada primero para la computación en tiempo real. La topología se enviará a un clúster. A continuación, un nodo máster del clúster distribuye códigos y asigna tareas a los nodos de trabajo. Una topología contiene dos funciones: spout and bolt. Un spout envía mensajes y envía flujos de datos en tuples. Un bolt convierte los flujos de datos y realiza operaciones de computación y filtrado. El bolt puede enviar datos aleatoriamente a otros bolts. Los tuples enviados por un spout son matrices incambiables y se asignan a pares de clave-valor fijos.

Figura 6-122 Arquitectura del sistema de Storm



La lógica de procesamiento de servicio está encapsulada en la topología de Storm. Una topología es un conjunto de componentes de spout (origen de datos) y bolt (procesamiento lógico) que se conectan mediante Stream Groupings en modo DAG. Todos los componentes (spout and bolt) de una topología trabajan en paralelo. En una topología, puede especificar el paralelismo para cada nodo. A continuación, Storm asigna tareas en el clúster para que la computación mejore las capacidades de procesamiento del sistema.

Figura 6-123 Topología



Storm es aplicable al análisis en tiempo real, la computación continua y la extracción, transformación y carga distribuida (ETL). Tiene las siguientes ventajas:

- Amplias aplicaciones
- Alta escalabilidad
- Cero pérdida de datos
- Alta tolerancia a fallos
- Fácil de construir y controlar
- Compatibilidad multilenguaje

Storm es una plataforma informática y proporciona Lenguaje de Consulta Continua (CQL) en la capa de servicio para facilitar la implementación del servicio. CQL tiene las siguientes características:

- Fácil de usar: La sintaxis CQL es similar a la sintaxis SQL. Los usuarios que tienen conocimientos básicos de SQL pueden aprender fácilmente CQL y usarlo para desarrollar servicios.
- Funciones ricas: Además de las expresiones básicas proporcionadas por SQL, CQL proporciona funciones, como ventanas, filtrado y configuración de simultaneidad, para el procesamiento de flujo.
- Fácil de escalar: CQL proporciona una API de extensión para soportar escenarios de servicio cada vez más complejos. Los usuarios pueden personalizar la entrada, la salida, la serialización y la deserialización para cumplir con los requisitos de servicio específicos.
- Fácil de depurar: CQL proporciona una explicación detallada de los códigos de error, facilitando a los usuarios la rectificación de fallas.

Para obtener más información sobre la arquitectura y los principios de Storm, consulte **https://storm.apache.org/**.

Principio

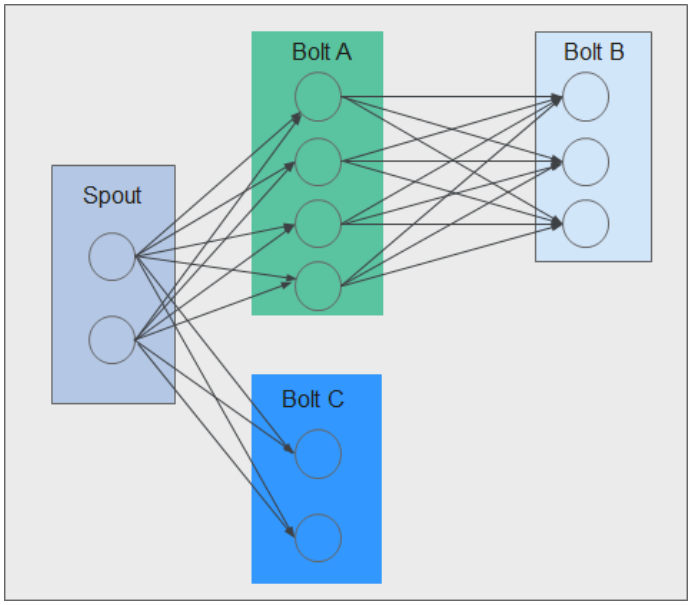
- **Conceptos básicos**

Tabla 6-26 Conceptos

Concepto	Descripción
Tuple	Un tuple es un par de clave-valor invariable utilizado para transferir datos. Tuples se crean y procesan de manera distribuida.
Stream	Un stream es una secuencia ilimitada de tuples.
Topology	Una topología es una aplicación en tiempo real que se ejecuta en la plataforma Storm. Es un gráfico acíclico dirigido (DAG) compuesto de componentes. Una topología puede ejecutarse simultáneamente en varias máquinas. Cada máquina ejecuta una parte del DAG. Una topología es similar a un MapReduce job. La diferencia es que la topología es un programa residente. Una vez iniciada, la topología no puede detenerse a menos que se termine manualmente.
Spout	Un spout es la fuente de tuples. Por ejemplo, un spout puede leer datos de una cola de mensajes, una base de datos, un sistema de archivos o una conexión TCP y los convierte en tuples, que son procesadas por el siguiente componente.
Bolt	En una topología, un bolt es un componente que recibe datos y ejecuta lógica específica, como filtrar o convertir tuples, unir o agregar flujos y realizar estadísticas y persistencia de resultados.
Worker	Un worker es un procesamiento físico en estado de ejecución en una topología. Cada worker es un proceso de JVM. Cada topología puede ser ejecutada por múltiples workers. Cada worker ejecuta un subconjunto lógico de la topología.
Task	Task es una rosca de spout o bolt de un worker.
Stream groupings	Stream grouping especifica las políticas de distribución de tuple. Instruye al bolt posterior cómo recibir los tuples. Las políticas admitidas incluyen Shuffle Grouping, Fields Grouping, All Grouping, Global Grouping, Non Grouping, y Directed Grouping.

Figura 6-124 muestra una topología (DAG) que consiste en un Spout and Bolt. En la figura, un rectángulo indica un Spout o Bolt, el nodo en cada rectángulo indica tareas, y las líneas entre tareas indican streams.

Figura 6-124 Topología



- **Confiabilidad**

  Storm ofrece tres niveles de confiabilidad de datos:

  - Como máximo una vez: Los datos procesados pueden perderse, pero no pueden procesarse repetidamente. Este nivel de confiabilidad ofrece el mayor rendimiento.
  - Al menos una vez: Los datos pueden procesarse repetidamente para garantizar una transmisión de datos confiable. Si no se recibe una respuesta dentro del tiempo especificado, el Spout vuelve a enviar los datos a Bolts para su procesamiento. Este nivel de confiabilidad puede afectar ligeramente al rendimiento del sistema.
  - Exactamente una vez: Los datos se transmiten con éxito sin pérdida o procesamiento de redundancia. Este nivel de confiabilidad ofrece el peor rendimiento.

  Seleccione el nivel de confiabilidad en función de los requisitos de servicio. Por ejemplo, para los servicios que requieren una alta confiabilidad de datos, utilice Exactamente una vez para asegurarse de que los datos se procesan solo una vez. Para los servicios insensibles a la pérdida de datos, utilice otros niveles para mejorar el rendimiento del sistema.

- **Tolerancia a fallas**

  Storm es un sistema tolerante a fallas que ofrece alta disponibilidad. **Tabla 6-27** describe la tolerancia a fallas de los componentes de Storm.

Tabla 6-27 Tolerancia a fallas

Escenario	Descripción
Nimbus failed	Nimbus es rápido y apátrida. Si el Nimbus activo es defectuoso, el Nimbus en espera se hace cargo de los servicios inmediatamente y proporciona servicios externos.

Escenario	Descripción
Supervisor failed	Supervisor es un daemon de fondo de Workers. Es fail-fast y apátrida. Si un Supervisor es defectuoso, los Workers que se ejecutan en el nodo no se ven afectados, pero no pueden recibir nuevas tareas. El OMS puede detectar la falla del Supervisor y reiniciar los procesos.
Worker failed	Si un trabajador está defectuoso, el supervisor del Worker lo reiniciará de nuevo. Si el reinicio falla varias veces, Nimbus reasigna tareas a otros nodos.
Node failed	Si un nodo está defectuoso, todas las tareas procesadas por el nodo expirarán y Nimbus asignará las tareas a otro nodo para su procesamiento.

Características de código abierto

- Cómputo en tiempo real distribuido

  En un clúster de Storm, cada máquina admite la ejecución de múltiples procesos de trabajo y cada proceso de trabajo puede crear múltiples subprocesos. Cada subproceso puede ejecutar varias tareas. Una tarea indica un procesamiento de datos simultáneo.
- Alta tolerancia a fallos

  Durante el procesamiento de mensajes, si un nodo o un proceso es defectuoso, la unidad de procesamiento de mensajes puede volver a desplegarse.
- Mensajes confiables

  Se admiten métodos de procesamiento de datos, incluidos At-Least Once, At-Most Once, y Exactly Once.
- Mecanismo de seguridad

  Storm proporciona autenticación basada en Kerberos y mecanismos de autorización conectables, admite SSL Storm UI y Log Viewer UI, y admite la integración de seguridad con otros componentes de la plataforma de big data (como ZooKeeper y HDFS).
- Definición e despliegue de topologías flexibles

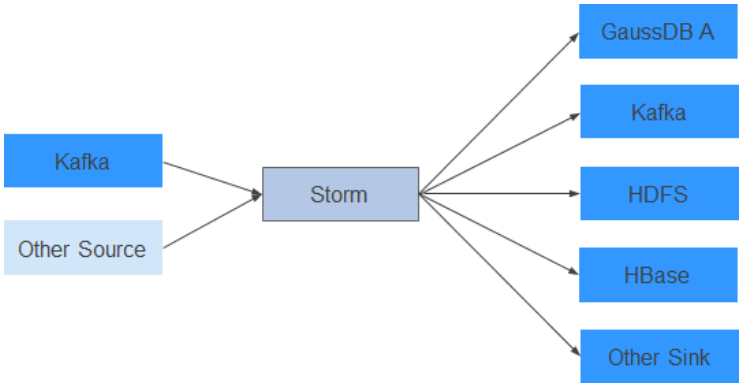
  El marco Flux se utiliza para definir e desplegar topologías de servicio. Si se cambia el DAG de servicio, los usuarios solo necesitan modificar YAML lenguaje específico del dominio (DSL), pero no necesitan volver a compilar o empaquetar código de servicio.
- Integración con componentes externos

  Storm admite la integración con múltiples componentes externos como Kafka, HDFS, HBase, Redis y JDBC/RDBMS, implementando servicios que involucran múltiples fuentes de datos.

6.29.2 Relación entre Storm y otros componentes

Storm proporciona un marco de computación distribuida en tiempo real. Puede obtener mensajes en tiempo real de fuentes de datos (como conexión de Kafka y TCP), realizar computación en tiempo real de alto rendimiento y baja latencia en una plataforma en tiempo real, y exportar resultados a colas de mensajes o implementar persistencia de datos. **Figura 6-125** muestra la relación entre Storm y otros componentes.

Figura 6-125 Relación con otros componentes



Relación entre Storm y Streaming

Tanto Storm como Streaming utilizan el kernel Apache Storm de código abierto. Sin embargo, la versión del kernel usada por Storm es 1.2.1 mientras que la usada por Streaming es 0.10.0. Streaming se utiliza para heredar servicios de transición en escenarios de actualización. Por ejemplo, si Streaming se ha desplegado en una versión anterior y los servicios se están ejecutando, Streaming puede seguir utilizándose después de la actualización. Se recomienda Storm en un nuevo clúster.

Storm 1.2.1 tiene las siguientes características nuevas:

- **Caché distribuida**: proporciona recursos externos (configuraciones) necesarios para compartir y actualizar la topología mediante herramientas de CLI. No es necesario volver a empaquetar y volver a desplegar la topología.
- **Native Streaming Window API**: proporciona las API basadas en ventanas.
- **Programador de recursos**: Agregó el complemento del programador de recursos. Al definir una topología, puede especificar el máximo de recursos disponibles y asignar cuotas de recursos a los usuarios, por lo tanto, para gestionar los recursos de topología de los usuarios.
- **State management**: proporciona a la API de Bolt el mecanismo de punto de control. Cuando un evento falla, Storm gestiona automáticamente el estado de Bolt y restaura el evento.
- **Muestreo y depuración de mensajes**: En la interfaz de usuario de Storm, puede habilitar o deshabilitar la depuración a nivel de topología o componente para enviar mensajes de flujo a registros especificados según la relación de muestreo.
- **Análisis dinámica de Worker**: En la interfaz de usuario de Storm, puede recopilar registros de jstack y heap del proceso de Worker y reiniciar el proceso de Worker.
- **Ajustes dinámicos de registros de topología**: Puede cambiar dinámicamente los registros de topología en ejecución en la interfaz de usuario de la CLI o Storm.
- **Rendimiento mejorado**: En comparación con las versiones anteriores, el rendimiento de Storm se ha mejorado mucho. Aunque el rendimiento de la topología está estrechamente relacionado con el caso de uso y la dependencia de servicios externos, el rendimiento es tres veces mayor en la mayoría de los escenarios.

6.29.3 Características mejoradas de código abierto de Storm

- CQL

El lenguaje de consulta continua (CQL) es un lenguaje similar a SQL utilizado para el procesamiento de flujo en tiempo real. En comparación con SQL, CQL ha introducido el concepto de ventana (secuenciación temporal), que permite almacenar y procesar datos en la memoria. La salida CQL es el resultado de cálculo de flujos de datos en un tiempo específico. El uso de CQL acelera el desarrollo de servicios, permite que las tareas se envíen fácilmente a la plataforma de Storm para su procesamiento en tiempo real, facilita la salida de resultados y permite que las tareas se terminen en el momento adecuado.

- Alta disponibilidad

  Nimbus HA garantiza el procesamiento continuo del servicio, como la adición de topologías y gestión, incluso si un Nimbus es defectuoso, lo que mejora la disponibilidad del clúster.

6.30 Tez

Tez es el último marco de computación de código abierto de Apache que soporta trabajos de gráficos acíclicos dirigidos (DAG). Puede convertir varios trabajos dependientes en un trabajo, lo que mejora en gran medida el rendimiento de los trabajos de DAG. Tez se basa en YARN y puede ejecutar trabajos de MapReduce sin ninguna modificación.

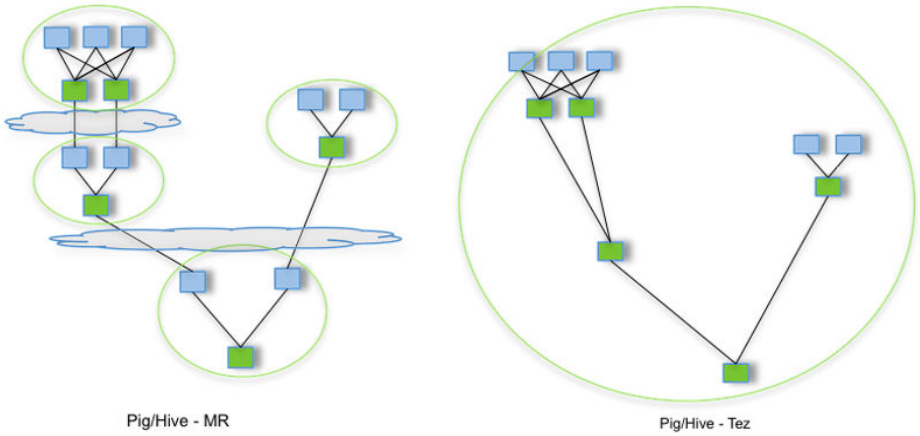
MRS utiliza Tez como motor de ejecución predeterminado de Hive. Tez supera notablemente al motor de computación MapReduce original en términos de eficiencia de ejecución.

Para más detalles sobre Tez, consulte **https://tez.apache.org/**.

Relación entre Tez y MapReduce

Tez utiliza un DAG para organizar tareas de MapReduce. En el DAG, un nodo es un RDD, y un borde indica una operación en el RDD. La idea principal es dividir aún más las tareas de Map y Reduce. Una tarea de Map se divide en las tareas Input-Processor-Sort-Merge-Output y la tarea Reduce se divide en las tareas Input-Shuffle-Sort-Merge-Process-output. Tez reagrupa de forma flexible varias tareas pequeñas para formar un trabajo DAG grande.

Figura 6-126 Procesos para enviar tareas usando Hive en el MapReduce y Hive en el Tez



Una tarea de Hive on MapReduce contiene varias tareas de MapReduce. Cada tarea almacena resultados intermedios en HDFS. El reducer en la etapa anterior proporciona datos para el mapper en la siguiente etapa. Una tarea Hive on Tez puede completar el mismo proceso de procesamiento en una sola tarea, y no es necesario acceder a HDFS entre tareas.

Relación entre Tez y Yarn

Tez es un marco de computación que se ejecuta en Yarn. El entorno de tiempo de ejecución se compone de ResourceManager y ApplicationMaster de Yarn. ResourceManager es un nuevo sistema de gestión de recursos, y ApplicationMaster es responsable de cortar los datos de trabajo de MapReduce, asignar tareas, solicitar recursos, programar tareas y tolerar fallas. Además, TezUI depende del TimelineServer proporcionado por Yarn para mostrar el proceso de ejecución de las tareas de Tez.

6.31 YARN

6.31.1 Principios básicos de YARN

La comunidad de código abierto de Apache presenta el marco de gestión de recursos unificado YARN para compartir clústeres de Hadoop, mejorar su escalabilidad y confiabilidad, y eliminar un cuello de botella de rendimiento de JobTracker en el marco de MapReduce inicial.

La idea fundamental de YARN es dividir las dos funcionalidades principales del JobTracker: la gestión de recursos y la programación/monitorización de trabajos, en daemons separados. La idea es tener un ResourceManager global (RM) y un ApplicationMaster por aplicación (AM).

NOTA

Una aplicación es un solo trabajo en el sentido clásico de trabajos de MapReduce o un gráfico acíclico dirigido (DAG) de trabajos.

Arquitectura

ResourceManager es la esencia de la estructura en capas de YARN. Esta entidad controla un clúster completo y gestiona la asignación de aplicaciones a los recursos informáticos subyacentes. El ResourceManager asigna cuidadosamente varios recursos (computación, memoria, ancho de banda, etc.) a los NodeManagers subyacentes (agentes por nodo de YARN). ResourceManager también trabaja con ApplicationMasters para asignar recursos, y trabaja con NodeManagers para iniciar y supervisar sus aplicaciones subyacentes. En este contexto, el ApplicationMaster ha tomado parte del papel del TaskTracker anterior, y el ResourceManager ha tomado el papel del JobTracker.

ApplicationMaster gestiona cada instancia de una aplicación que se ejecuta en YARN. ApplicationMaster negocia recursos desde ResourceManager y trabaja con NodeManagers para supervisar la ejecución de contenedores y el uso de recursos (asignación de recursos de memoria y CPU).

El NodeManager gestiona cada nodo de un clúster de YARN. El NodeManager proporciona servicios por nodo en un clúster, desde la supervisión de la gestión de un contenedor a lo largo de su ciclo de vida hasta la supervisión de los recursos y el seguimiento del estado de sus nodos. MRv1 gestiona la ejecución de las tareas Map y Reduce a través de ranuras, mientras que NodeManager gestiona contenedores abstractos, que representan recursos por nodo disponibles para una aplicación en particular.

Figura 6-127 Arquitectura

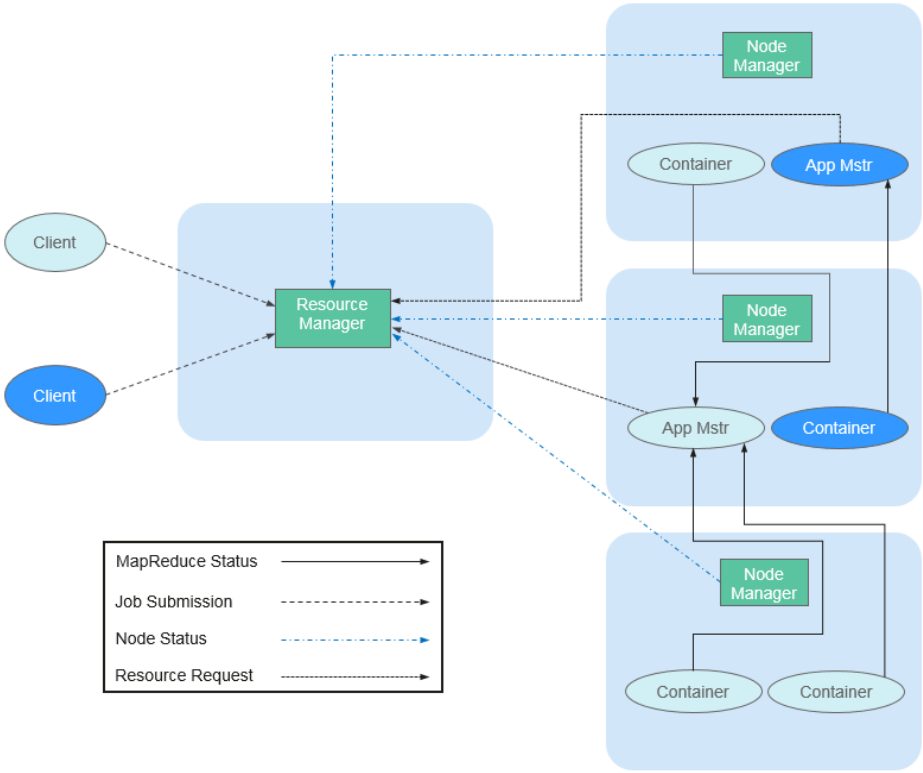


Tabla 6-28 describe los componentes mostrados en **Figura 6-127**.

Tabla 6-28 Descripción de la arquitectura

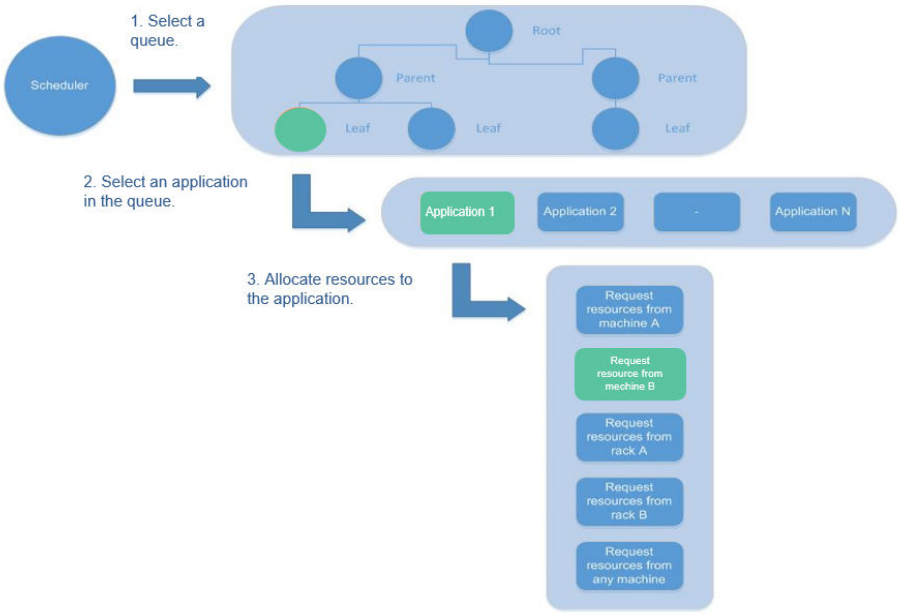
Nombre	Descripción
Client	Cliente de un YARN application. Puede enviar una tarea a ResourceManager y consultar el estado operativo de una application mediante el cliente.
ResourceManager(RM)	RM gestiona y asigna de forma centralizada todos los recursos del clúster. Recibe información de informes de recursos de cada nodo (NodeManager) y asigna recursos a aplicaciones basándose en los recursos recopilados de acuerdo con una política especificada.
NodeManager(NM)	NM es el agente en cada nodo de YARN. Gestiona el nodo informático en el clúster Hadoop, establece comunicación con ResourceManger; monitoriza el ciclo de vida de containers; monitoriza el uso de recursos como la memoria y la CPU de cada container; rastrea el estado del nodo y gestiona los registros y servicios auxiliares utilizados por diferentes aplicaciones.

Nombre	Descripción
Application Master(AM)	AM (App Mstr en la figura anterior) es responsable de todas las tareas a través del ciclo de vida de un application. Las tareas incluyen las siguientes: Negociar con un planificador RM para obtener un recurso; asignar adicionalmente los recursos obtenidos a tareas internas (asignación secundaria de recursos); comunicarse con el NM para iniciar o detener tareas; supervisar el estado de ejecución de todas las tareas; y volver a aplicar recursos para las tareas para reiniciar las tareas cuando las tareas no se ejecuten.
Container	Una abstracción de recursos en YARN. Encapsula recursos multidimensionales (incluyidos solo memoria y CPU) en un nodo determinado. Cuando ApplicationMaster se aplica a recursos de ResourceManager, ResourceManager devuelve recursos a ApplicationMaster en un container. YARN asigna un container para cada tarea y la tarea solo puede usar los recursos encapsulados en el container.

En YARN, programadores de recursos organizan los recursos mediante colas jerárquicas. Esto garantiza que los recursos se asignen y compartan entre las colas, mejorando así el uso de los recursos del clúster. El modelo de asignación de recursos básicos de Superior Scheduler es el mismo que el de Capacity Scheduler, como se muestra en la siguiente figura.

Un programador mantiene la información de la cola. Puede enviar solicitudes a una o más colas. Durante cada latido de NM, el programador selecciona una cola de acuerdo con una regla de planificación específica, selecciona una aplicación en la cola y, a continuación, asigna recursos a la aplicación. Si los recursos no se pueden asignar a la aplicación debido al límite de algunos parámetros, el programador seleccionará otra aplicación. Después de la selección, el programador procesa la solicitud de recursos de esta aplicación. El programador da prioridad a las solicitudes de recursos locales primero, luego para recursos en el mismo rack y, finalmente, para recursos de cualquier máquina.

Figura 6-128 Modelo de asignación de recursos



Principio

El nuevo marco de Hadoop MapReduce se denomina MRv2 o YARN. YARN consta de ResourceManager, ApplicationMaster y NodeManager.

- ResourceManager es un gestor de recursos global que gestiona y asigna recursos en el sistema. ResourceManager se compone de Scheduler y Applications Manager.
  - El programador asigna recursos del sistema a todas las aplicaciones en ejecución según las restricciones, como la capacidad y la cola (por ejemplo, asigna una cierta cantidad de recursos para una cola y ejecuta un número específico de trabajos). Asigna recursos en función de la demanda de las aplicaciones, utilizándose el container como unidad de asignación de recursos. Funcionando como una unidad de asignación de recursos dinámicos, Container encapsula recursos de memoria, CPU, disco y red, limitando así el recurso consumido por cada tarea. Además, el programador es un componente conectable. Puede diseñar nuevos programadores según sea necesario. YARN proporciona varios programadores directamente disponibles, como Fair Scheduler y Capacity Scheduler.
  - Applications Manager gestiona todas las aplicaciones del sistema e implica enviar aplicaciones, negociar con programadores sobre recursos, habilitar y supervisar ApplicationMaster y reiniciar ApplicationMaster cuando falla el inicio.
- NodeManager es el gestor de recursos y tareas de cada nodo. Por un lado, el NodeManager informa periódicamente a ResourceManager el uso de recursos del nodo local y el estado de ejecución de cada Container. Por otro lado, el NodeManager recibe y procesa solicitudes de ApplicationMaster para iniciar o detener Containers.
- ApplicationMaster es responsable de todas las tareas durante el ciclo de vida de una aplicación, estos canales incluyen lo siguiente:
  - Negociar con el programador de RM para obtener recursos.
  - Asignar recursos a componentes internos (asignación secundaria de recursos).
  - Comunicar con NodeManager para iniciar o detener tareas.
  - Supervisar el estado de ejecución de todas las tareas y vuelver a solicitar recursos para las tareas cuando las tareas no se ejecuten para reiniciarlas.

Principio de Capacity Scheduler

Capacity Scheduler es un programador multiusuario. Asigna recursos por cola y establece los recursos mínimos/máximos que se pueden usar para cada cola. Además, el límite superior de uso de recursos se establece para cada usuario para evitar el abuso de recursos. Los recursos restantes de una cola se pueden compartir temporalmente con otras colas.

Capacity Scheduler admite múltiples colas. Configura una cierta cantidad de recursos para cada cola y adopta la política de programación de la cola primero en entrar primero en salir (FIFO). Para evitar que las aplicaciones de un usuario utilicen exclusivamente los recursos de una cola, Capacity Scheduler establece un límite en el número de recursos utilizados por los trabajos enviados por un usuario. Durante la programación, Capacity Scheduler calcula primero el número de recursos necesarios para cada cola y selecciona la cola que requiere el menor número de recursos. A continuación, asigna recursos según la prioridad del trabajo y el tiempo en que se envían los trabajos, así como el límite de recursos y memoria. Capacity Scheduler admite las siguientes características:

- Capacidad garantizada: como administrador del clúster MRS, puede establecer los límites inferior y superior del uso de recursos para cada cola. Todas las aplicaciones enviadas a esta cola comparten los recursos.

- Alta flexibilidad: temporalmente, los recursos restantes de una cola se pueden compartir con otras colas. Sin embargo, dichos recursos deben liberarse en caso de que se envíe una nueva aplicación a la cola. Esta asignación flexible de recursos ayuda a mejorar notablemente el uso de los recursos.
- Multitenancy: Varios usuarios pueden compartir un clúster y varias aplicaciones pueden ejecutarse simultáneamente. Para evitar el uso exclusivo de recursos por una sola aplicación, usuario o cola, el administrador del clúster de MRS puede agregar varias restricciones (por ejemplo, limitar las tareas simultáneas de una sola aplicación).
- Protección asegurada: Se proporciona una lista de ACL para cada cola para limitar estrictamente el acceso del usuario. Puede especificar los usuarios que pueden ver el estado de su aplicación o controlar las aplicaciones. Además, el administrador del clúster MRS puede especificar un administrador de colas y un administrador del sistema del clúster.
- Actualización dinámica de los archivos de configuración: los administradores de clústeres MRS pueden modificar dinámicamente los parámetros de configuración para gestionar clústeres en línea.

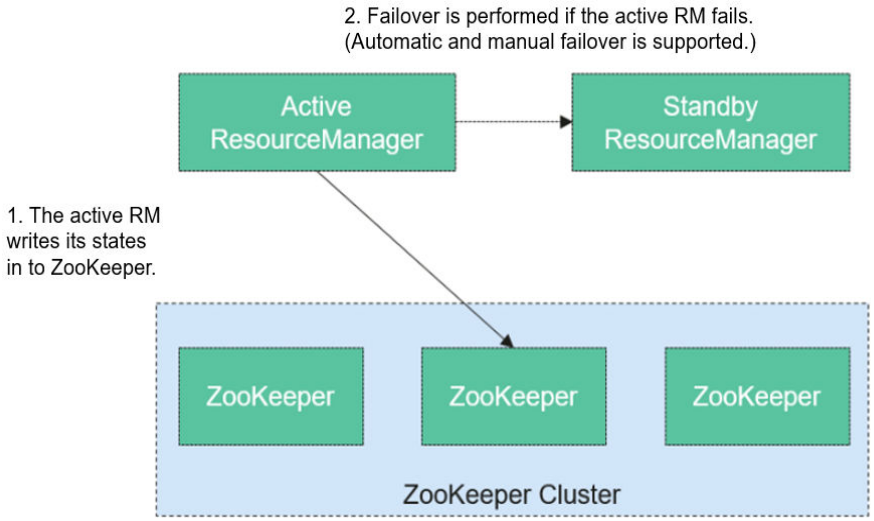
Cada cola en Capacity Scheduler puede limitar el uso de recursos. Sin embargo, el uso de recursos de una cola determina su prioridad cuando se asignan recursos a colas, lo que indica que las colas con menor capacidad son competitivas. Si el rendimiento de un clúster es grande, la programación de retardo permite a una aplicación renunciar a la programación entre máquinas o entre bastidores, y solicitar la planificación local.

6.31.2 Solución de YARN HA

Principios de HA y solución de implementación

ResourceManager en YARN gestiona los recursos y programa las tareas en el clúster. En versiones anteriores a Hadoop 2.4, los SPOF pueden ocurrir en el ResourceManager en el clúster de YARN. La solución YARN HA utiliza nodos de ResourceManager redundantes para hacer frente a los desafíos de confiabilidad del servicio y tolerancia a fallas.

Figura 6-129 Arquitectura de HA de ResourceManager



El HA de ResourceManager se consigue usando nodos de ResourceManager activo-en espera, como se muestra en **Figura 6-129**. Al igual que la solución HDFS HA, el HA

ResourceManager permite que solo un nodo de ResourceManager esté en el estado activo en cualquier momento. Cuando el ResourceManager activo falla, la conmutación activa-en espera se puede activar de forma automática o manual.

Cuando la función de conmutación por error automática no está habilitada, después de que el clúster YARN esté habilitado, los administradores de clúster MRS necesitan ejecutar el comando ***yarn rmadmin*** para cambiar manualmente uno de los nodos ResourceManager al estado activo. Tras un evento de mantenimiento planificado o una falla, se espera que primero desciendan el ResourceManager activo al estado de espera y el ResourceManager de espera promocionen al estado activo.

Cuando se habilita la conmutación por error automática, se utiliza un ActiveStandbyElector integrado basado en ZooKeeper para decidir qué nodo ResourceManager debe ser el activo. Cuando el ResourceManager activo es defectuoso, se selecciona automáticamente otro nodo ResourceManager para que sea el activo para hacerse cargo del nodo defectuoso.

Cuando los nodos ResourceManager del clúster se implementan en modo HA, el **yarn-site.xml** de configuración utilizado por los clientes debe enumerar todos los nodos de ResourceManager. El cliente (incluido ApplicationMaster y NodeManager) busca el ResourceManager activo en modo de sondeo. Es decir, el cliente necesita proporcionar el mecanismo de tolerancia a fallas. Si no se puede conectar con el ResourceManager activo, el cliente busca continuamente uno nuevo en modo de sondeo.

Después de que el nodo ResourceManager en espera se convierte en el activo, las aplicaciones de capa superior pueden recuperarse a su estado cuando se produce el fallo. Para obtener más información, consulte **Reiniciar ResourceManager**. Cuando ResourceManager Restart está habilitado, el nodo ResourceManager reiniciado carga la información del nodo activo anterior de ResourceManager y toma la información del estado del contenedor en todos los nodos de NodeManager para continuar la ejecución del servicio. De esta manera, la información de estado se puede guardar ejecutando periódicamente operaciones de punto de control, evitando la pérdida de datos. Asegúrese de que los nodos de ResourceManager activos y en espera puedan acceder a la información de estado. Actualmente, se proporcionan tres métodos para compartir información de estado por sistema de archivos (FileSystemRMStateStore), base de datos de LevelDB (LeveldbRMStateStore) y ZooKeeper (ZKRMStateStore). Entre ellos, solo ZKRMStateStore es compatible con el mecanismo de Fencing. De forma predeterminada, Hadoop utiliza ZKRMStateStore.

Para obtener más información acerca de la solución YARN HA, visite el siguiente sitio Web:

http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

https://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

6.31.3 Relación entre YARN y otros componentes

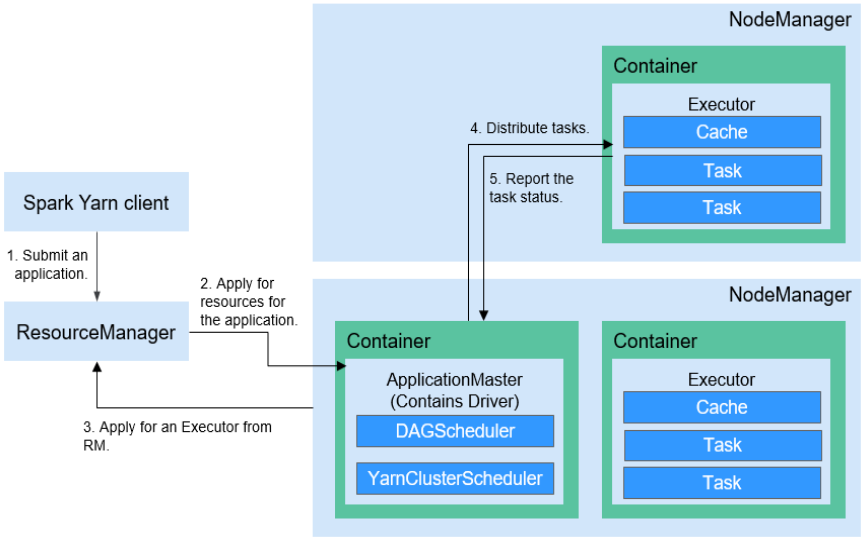
Relación entre YARN y Spark

La computación y programación de Spark pueden implementarse usando el modo YARN. Spark disfruta de los recursos informáticos proporcionados por los clústeres de YARN y ejecuta tareas de forma distribuida. Spark en YARN tiene dos modos: YARN-cluster y YARN-client.

- Modo de YARN Cluster

  Figura 6-130 describe el marco de operación.

Figura 6-130 Marco de operación de Spark on YARN-cluster



Proceso de implementación de Spark on YARN-cluster:

a. El cliente genera la información de la aplicación y, a continuación, envía la información a ResourceManager.

b. ResourceManager asigna el primer container (ApplicationMaster) a SparkApplication e inicia el driver en el container.

c. ApplicationMaster se aplica a los recursos de ResourceManager para ejecutar el container.

   ResourceManager asigna los containers a ApplicationMaster, que se comunica con los NodeManagers relacionados e inicia el executor en el container obtenido. Después de iniciar el executor, se registra con drivers y se aplica a las tareas.

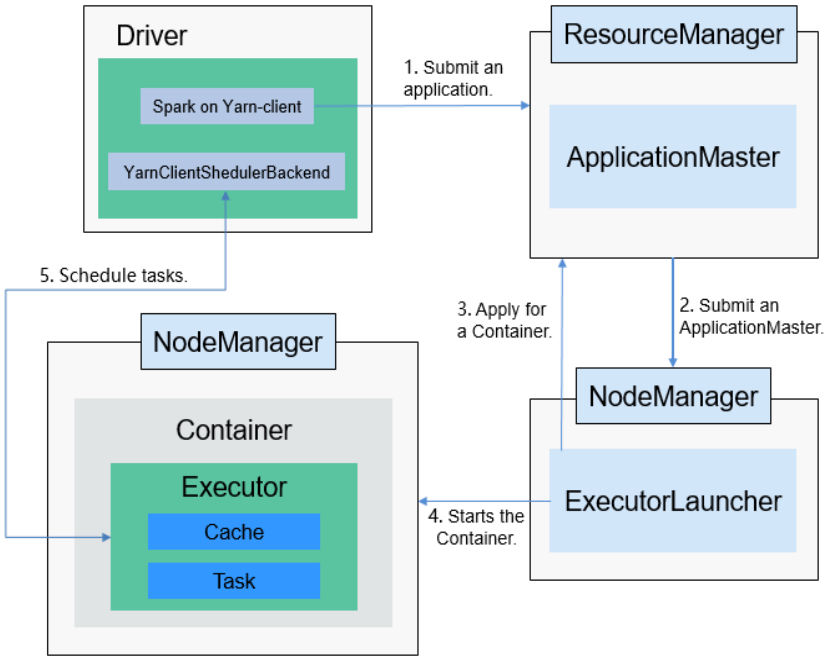
d. Los controladores asignan tareas a los ejecutores.

e. Los executors ejecutan tareas e informan el estado de funcionamiento a Drivers.

- Modo de YARN Client

  Figura 6-131 describe el marco de operación.

Figura 6-131 Marco de operación de Spark on YARN-client



Proceso de implementación de Spark on YARN-client:

NOTA

En el modo YARN-cliente, el controlador se implementa y se inicia en el cliente. En el modo YARN-client, el cliente de una versión anterior es incompatible. Se recomienda utilizar el modo YARN-cluster.

a. El cliente envía la solicitud de aplicación de Spark a ResourceManager y, a continuación, ResourceManager devuelve los resultados. Los resultados incluyen información como Application ID y los recursos máximos y mínimos disponibles. El cliente empaqueta toda la información necesaria para iniciar ApplicationMaster y envía la información a ResourceManager.

b. Después de recibir la solicitud, ResourceManager encuentra un nodo apropiado para ApplicationMaster y lo inicia en este nodo. ApplicationMaster es un papel en YARN, y el nombre del proceso en Spark es ExecutorLauncher.

c. En función de los requerimientos de recursos de cada tarea, ApplicationMaster puede solicitar una serie de containers para ejecutar tareas desde ResourceManager.

d. Después de recibir la lista de container recién asignados (de ResourceManager), ApplicationMaster envía información a los NodeManagers relacionados para iniciar los containers.

   ResourceManager asigna los containers a ApplicationMaster, que se comunica con los NodeManagers relacionados e inicia el executor en el container obtenido. Después de iniciar el executor, se registra con drivers y se aplica a las tareas.

   NOTA

   Los containers en ejecución no se suspenden y los recursos no se liberan.

e. Drivers asignan task a los executors. Los executors ejecutan tareas e informan el estado de funcionamiento a Drivers.

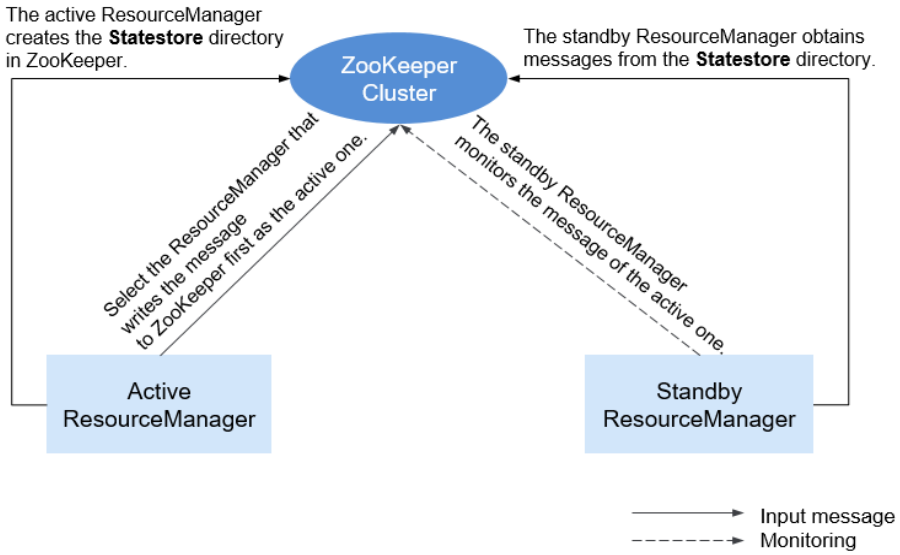
Relación entre YARN y MapReduce

MapReduce es un marco de computación que se ejecuta en YARN, que se utiliza para el procesamiento por lotes. MRv1 se implementa basado en MapReduce en Hadoop 1.0, que se compone de modelos de programación (Las API de programación nuevas y antiguas), entorno de ejecución (JobTracker y TaskTracker), y motor de procesamiento de datos (MapTask y ReduceTask). Este marco sigue siendo débil en escalabilidad, tolerancia a fallas (JobTracker SPOF) y compatibilidad con varios marcos. (Actualmente, solo se admite el marco de computación de MapReduce.) MRv2 se implementa en base a MapReduce en Hadoop 2.0. El código fuente reutiliza los modelos de programación MRv1 y la implementación del motor de procesamiento de datos, y el entorno de ejecución se compone de ResourceManager y ApplicationMaster. ResourceManager es un nuevo sistema de gestión de recursos, y ApplicationMaster es responsable de cortar los datos de trabajo de MapReduce, asignar tareas, solicitar recursos, programar tareas y tolerar fallos.

Relación entre YARN y ZooKeeper

Figura 6-132 muestra la relación entre ZooKeeper y YARN.

Figura 6-132 Relación entre ZooKeeper y YARN



1. Cuando se inicia el sistema, ResourceManager intenta escribir información de estado en el ZooKeeper. ResourceManager que escribe primero información de estado en ZooKeeper se selecciona como ResourceManager activo, y otros son ResourceManagers en espera. Los ResourceManagers en espera monitorean periódicamente la información de elección activa de ResourceManager en ZooKeeper.
2. El ResourceManager activo crea el directorio **Statestore** de ZooKeeper para almacenar la información de la aplicación. Si el ResourceManager activo es defectuoso, el ResourceManager en espera obtiene la información de la aplicación del directorio **Statestore** y restaura los datos.

Relación entre YARN y Tez

La información del trabajo de Hive en Tez requiere la capacidad del servidor de TimeLine de YARN para que las tareas de Hive puedan mostrar el estado actual e histórico de las aplicaciones, facilitando el almacenamiento y la recuperación.

6.31.4 Funciones de código abierto mejoradas de Yarn

Programación de tareas basada en prioridades

En el mecanismo nativo de programación de recursos de Yarn, si todos los recursos del clúster de Hadoop están ocupados por los trabajos de MapReduce enviados anteriormente, los trabajos enviados más tarde se mantendrán en estado pendiente hasta que se ejecuten todos los trabajos en ejecución y se liberen los recursos.

El clúster MRS proporciona el mecanismo de programación de prioridades de tareas. Con esta función, puede definir trabajos de diferentes prioridades. Los trabajos de alta prioridad pueden adelantarse a los recursos liberados de trabajos de baja prioridad aunque los trabajos de alta prioridad se envíen más tarde. Los trabajos de baja prioridad que no se inician se suspenderán a menos que los trabajos de alta prioridad se completen y se liberen recursos, entonces se puedan iniciar correctamente.

Esta característica permite que los servicios controlen los trabajos informáticos de manera más flexible, logrando así una mayor utilización de los recursos del clúster.

NOTA

La reutilización de contenedores entra en conflicto con la programación de prioridades de tareas. Si se habilita la reutilización de contenedores, se están ocupando recursos y la programación de prioridades de tareas no tiene efecto.

Control de permisos de Yarn

El mecanismo de permiso de Hadoop Yarn se implementa a través de ACL. A continuación se describe cómo conceder diferentes controles de permisos a diferentes usuarios:

- Admin ACL

  Se especifica un administrador de O&M para el clúster YARN. **yarn.admin.acl** determina la ACL Admin. El administrador del clúster O&M puede acceder a la interfaz de usuario web de ResourceManager y operar nodos de NodeManager, colas y NodeLabel, **pero no puede enviar tareas**.

- Queue ACL

  Para facilitar la gestión de usuarios en el clúster, los usuarios o grupos de usuarios se dividen en varias colas a las que pertenece cada usuario y grupo de usuarios. Cada cola contiene permisos para enviar y gestionar aplicaciones (por ejemplo, terminar cualquier aplicación).

Funciones de código abierto:

Actualmente, Yarn admite los siguientes roles para los usuarios:

- Administrador de Clúster O&M
- Administrador de colas
- Usuario común

Sin embargo, las APIs (como la interfaz de usuario web, la API de REST y la API de Java) proporcionado por Yarn no admite el control de permisos específicos de función. Por lo tanto, todos los usuarios tienen el permiso para acceder a la información de la aplicación y del clúster, que no cumple con los requisitos de aislamiento en el escenario de multitenant.

Esta es una función mejorada.

En el modo de seguridad, la gestión de permisos se mejora para las API, como la interfaz de usuario web, la API de REST y la API de Java proporcionada por Yarn. El control de permisos se puede realizar en función de los roles de usuario.

Los permisos basados en roles son los siguientes:

- Administrador de clúster O&M: realiza operaciones de gestión en el clúster de Yarn, como acceder a la interfaz de usuario web ResourceManager, actualizar colas, configurar NodeLabel y realizar conmutación activa/en espera.
- Administrador de colas: tiene el permiso para modificar y ver colas gestionadas por el clúster Yarn.
- Usuario común: tiene el permiso para modificar y ver las aplicaciones autoenviadas en el clúster de Yarn.

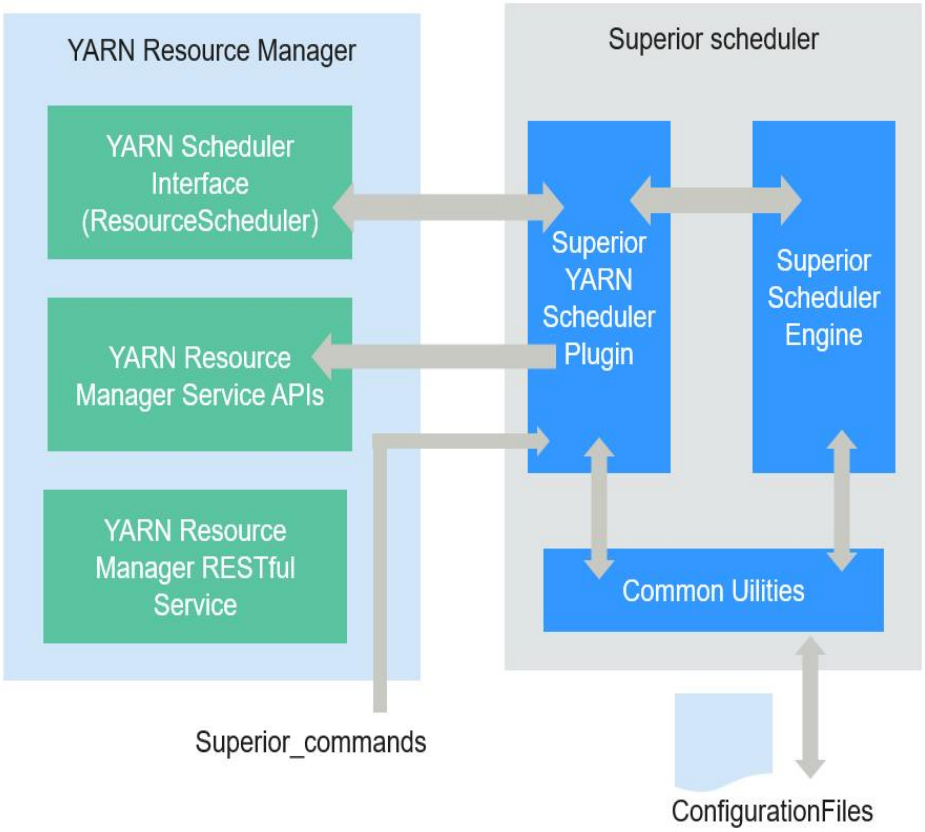
Principio del programador Superior (autodesarrollado)

Superior Scheduler es un motor de programación diseñado para el sistema de gestión de recursos distribuidos Hadoop Yarn. Es un programador de alto rendimiento y de nivel empresarial diseñado para grupos de recursos convergentes y requisitos de servicio de inquilinos múltiples.

Superior Scheduler logra todas las funciones de programadores de código abierto, Fair Scheduler y Capacity Scheduler. En comparación con los programadores de código abierto, Superior Scheduler está mejorado en la política de programación de recursos multitenant empresarial, el aislamiento de recursos y el uso compartido entre los usuarios de un inquilino, el rendimiento de planificación, el uso de recursos del sistema y la escalabilidad del clúster. Superior Scheduler está diseñado para reemplazar a los programadores de código abierto.

Al igual que Fair Scheduler de código abierto y Capacity Scheduler, Superior Scheduler sigue la API del complemento de programador de Yarn para interactuar con el ResourceManager de Yarn para ofrecer funcionalidades de programación de recursos. **Figura 6-133** muestra el diagrama general del sistema.

Figura 6-133 Arquitectura interna del Superior Scheduler



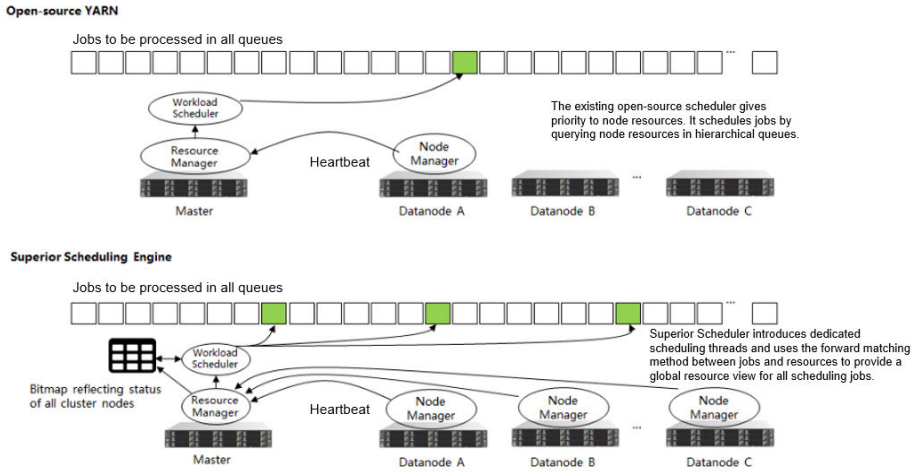
En **Figura 6-133**, Superior Scheduler consta de los siguientes módulos:

- Superior Scheduler Engine es un motor de programación de alto rendimiento con políticas de programación ricas.
- El complemento de Superior Yarn Scheduler funciona como un puente entre el Yarn ResourceManager y Superior Scheduler Engine e interactúa con Yarn ResourceManager.

  El principio de programación de los programadores de código abierto es que los recursos coinciden con los trabajos basados en los latidos de los nodos informáticos. Específicamente, cada nodo informático envía periódicamente mensajes de latidos al ResourceManager de Yarn para notificar el estado del nodo e inicia el programador para asignar trabajos al propio nodo. En este mecanismo de programación, el período de programación depende del latido del corazón. Si aumenta la escala del clúster, puede producirse un cuello de botella en la escalabilidad del sistema y el rendimiento de programación. Además, debido a que los recursos coinciden con los trabajos, la precisión de programación de un programador de código abierto es limitada. Por ejemplo, la afinidad de datos es aleatoria y el sistema no admite políticas de programación basadas en carga. Es posible que el programador no tome la mejor decisión debido a la falta de la vista global de recursos al seleccionar trabajos.

  Superior Scheduler adopta múltiples mecanismos de programación. Hay subprocesos de programación dedicados en Superior Scheduler, separando los latidos con la programación y la prevención de tormentas de latidos del sistema. Además, Superior Scheduler hace coincidir los trabajos con los recursos, proporcionando a cada trabajo programado una vista global de recursos y aumentando la precisión de la programación. En comparación con el programador de código abierto, Superior Scheduler sobresale en rendimiento del sistema, uso de recursos y afinidad de datos.

Figura 6-134 Comparación de Superior Scheduler con programadores de código abierto



Aparte del rendimiento y la utilización mejorados del sistema, Superior Scheduler proporciona las siguientes características de programación principales:

- Varios grupos de recursos

  Varios grupos de recursos ayudan a dividir lógicamente los recursos del clúster y compartirlos entre varios tenants o colas. La división de grupos de recursos admite recursos heterogéneos. Los grupos de recursos se pueden dividir exactamente de acuerdo con los requisitos sobre el aislamiento de recursos de la aplicación. Puede configurar más políticas para diferentes colas en un grupo.

- Programación multitenant (**reserve**, **min**, **share** y **max**) en cada grupo de recursos

  Superior Scheduler proporciona una política flexible de programación jerárquica multitenant. Se pueden configurar diferentes políticas para diferentes tenants o colas que pueden acceder a diferentes grupos de recursos. En la siguiente figura se enumeran las políticas admitidas:

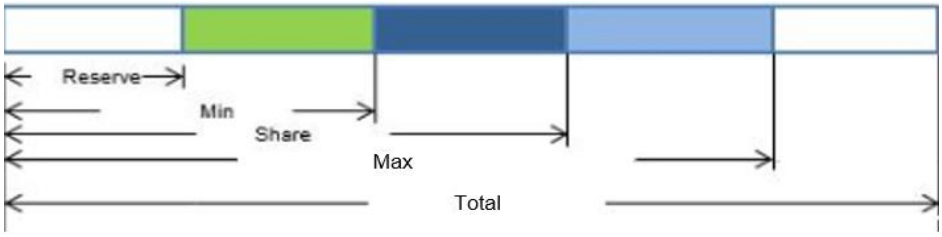
  Tabla 6-29 Descripción de la política

Nombre	Descripción
reserve	Esta política se utiliza para reservar recursos para un tenant. Aunque tenant no tiene trabajos disponibles, otro tenant no puede usar el recurso reservado. El valor puede ser un porcentaje o un valor absoluto. Si se configuran tanto el porcentaje como el valor absoluto, el porcentaje se calcula automáticamente en un valor absoluto y se utiliza el valor mayor. El valor predeterminado de **reserve** es **0**. En comparación con el método de especificar un grupo de recursos dedicado y hosts, la política **reserve** proporciona una función de reserva flotante flexible. Además, debido a que no se especifican hosts específicos, se mejora la afinidad de los datos para el cálculo y se evita el impacto de los hosts defectuosos.

Nombre	Descripción
min	Esta política permite la preferencia de recursos mínimos. Otros tenants pueden usar estos recursos, pero el tenant actual tiene la prioridad de usarlos. El valor puede ser un porcentaje o un valor absoluto. Si se configuran tanto el porcentaje como el valor absoluto, el porcentaje se calcula automáticamente en un valor absoluto y se utiliza el valor mayor. El valor predeterminado es **0**.
share	Esta política se utiliza para los recursos compartidos que no se pueden anular. Para usar estos recursos, el tenant actual debe esperar a que otros tenants completen trabajos y liberen recursos. El valor puede ser un porcentaje o un valor absoluto.
max	Esta política se utiliza para el máximo de recursos que se pueden utilizar. El tenant no puede obtener más recursos que el valor máximo permitido. El valor puede ser un porcentaje o un valor absoluto. Si se configuran tanto el porcentaje como el valor absoluto, el porcentaje se calcula automáticamente en un valor absoluto y se utiliza el valor mayor. Por valor predeterminado, no hay ninguna restricción en los recursos.

Figura 6-135 muestra la política de asignación de recursos del tenant.

Figura 6-135 Políticas de programación de recursos



NOTA

En la figura anterior, **Total** indica el número total de recursos, no la política de programación.

En comparación con los programadores de código abierto, Superior Scheduler soporta tanto el porcentaje como el valor absoluto de los tenants para asignar recursos, abordando de manera flexible los requisitos de programación de recursos de los tenants de nivel empresarial. Por ejemplo, los recursos se pueden asignar de acuerdo con el valor absoluto de los tenants de nivel 1, evitando el impacto causado por los cambios de escala de clúster. Sin embargo, los recursos se pueden asignar de acuerdo con el porcentaje de asignación de subtenants, mejorando los usos de recursos en el tenant de nivel 1.

- Programación de recursos heterogénea y multidimensional

  Superior Scheduler admite las siguientes funciones, excepto la programación de CPU y memoria:

  - Las etiquetas de nodo se pueden usar para identificar atributos multidimensionales de nodos como **GPU_ENABLED** y **SSD_ENABLED** y se pueden programar basándose en estas etiquetas.

  - Los grupos de recursos se pueden utilizar para agrupar recursos del mismo tipo y asignarlos a tenants o colas específicos.

- Programación justa de múltiples usuarios en un tenant

  En un leaf tenant, varios usuarios pueden usar la misma cola para enviar trabajos. En comparación con los programadores de código abierto, Superior Scheduler admite la configuración de políticas flexibles de uso compartido de recursos entre diferentes usuarios en un mismo tenant. Por ejemplo, los usuarios VIP pueden configurarse con mayor ponderación de acceso a recursos.

- Programación con conocimiento de localidad de datos

  Superior Scheduler adopta la política de programación de trabajo a nodo. Es decir, Superior Scheduler intenta programar trabajos especificados entre nodos disponibles para que el nodo seleccionado sea adecuado para los trabajos especificados. Al hacerlo, el programador tendrá una vista general del clúster y los datos. La localización está asegurada si existe la oportunidad de colocar las tareas más cerca de los datos. El programador de código abierto utiliza la política de programación de nodo a trabajo para hacer coincidir los trabajos apropiados con un nodo determinado.

- Reserva dinámica de recursos durante la programación de container

  En un entorno informático heterogéneo y diversificado, algunos containers necesitan más recursos o múltiples recursos. Por ejemplo, el trabajo Spark puede requerir una gran memoria. Cuando tales containers compiten con containers que requieren menos recursos, los containers que requieren más recursos pueden no obtener recursos suficientes dentro de un período razonable. Los programadores de código abierto asignan recursos a trabajos, lo que puede provocar una reserva de recursos no razonable para estos trabajos. Este mecanismo conduce al desperdicio de recursos generales del sistema. Superior Scheduler difiere de los programadores de código abierto en los siguientes aspectos:

  - Coincidencia basada en requisitos: Superior Scheduler programa los trabajos a los nodos y selecciona los nodos apropiados para reservar recursos para mejorar el tiempo de inicio de los contenedores y evitar el desperdicio.
  - Reequilibrio de tenant: cuando la lógica de reserva está habilitada, los programadores de código abierto no cumplen con la política de uso compartido configurada. Superior Scheduler utiliza diferentes métodos. En cada período de programación, Superior Scheduler recorre todos los tenants e intenta equilibrar los recursos según la política de multitenant. Además, el programador superior intenta cumplir con todas las políticas (**reserve**, **min** y **share**) para liberar recursos reservados y dirigir los recursos disponibles a otros containers que deben obtener recursos bajo diferentes tenants.

- Control dinámico del estado de la cola (**Open**/**Closed**/**Active**/**Inactive**)

  Se admiten varios estados de cola, lo que ayuda a los administradores de clústeres de MRS a gestionar y mantener varios tenants.

  - Estado abierto (**Open/Closed**): si el estado es **Open** de forma predeterminada, se aceptan las aplicaciones enviadas a la cola. Si el estado es **Closed**, no se acepta ninguna aplicación.
  - Estado activo (**Active/Inactive**): si el estado es **Active** de forma predeterminada, los recursos se pueden programar y asignar a las aplicaciones en el tenant. Los recursos no se programarán en colas en estado **Inactive**.

- Motivo pendiente de solicitud

  Si la solicitud no se ha iniciado, indique las razones pendientes del trabajo.

Tabla 6-30 describe el resultado de la comparación de programadores de código abierto de Superior Scheduler y Yarn.

Tabla 6-30 Análisis comparativo

Programación	Programador de código abierto de Yarn	Superior Scheduler
Programación multitenant	En clústeres homogéneos, se puede seleccionar Capacity Scheduler o Fair Scheduler y el clúster no es compatible con Fair Scheduler. Capacity Scheduler admite la programación por porcentaje y Fair Scheduler admite la programación por valor absoluto.	● Admite clústeres heterogéneos y múltiples grupos de recursos. ● Soporta **reservation** para garantizar el acceso directo a los recursos.
Programación con conocimiento de localidad de datos	La política de planificación de nodo a trabajo reduce la tasa de éxito de la localización de datos y potencialmente afecta al rendimiento de la ejecución de la aplicación.	**job-to-node scheduling policy** puede conocer la ubicación de los datos con mayor precisión, y la tasa de aciertos del trabajo de la programación de localización de datos es mayor.
Programación equilibrada basada en la carga de hosts	No compatible	**Se puede lograr una programación equilibrada cuando Superior Scheduler considera la carga del host y la asignación de recursos durante la programación.**
Programación justa de múltiples usuarios en un tenant	No compatible	Soporta palabras clave **default** y **others**.
Razón de espera de trabajo	No compatible	Las razones de espera de trabajo ilustran por qué un trabajo necesita esperar.

En conclusión, Superior Scheduler es un programador de alto rendimiento con varias políticas de programación y es mejor que Capacity Scheduler en términos de funcionalidad, rendimiento, uso de recursos y escalabilidad.

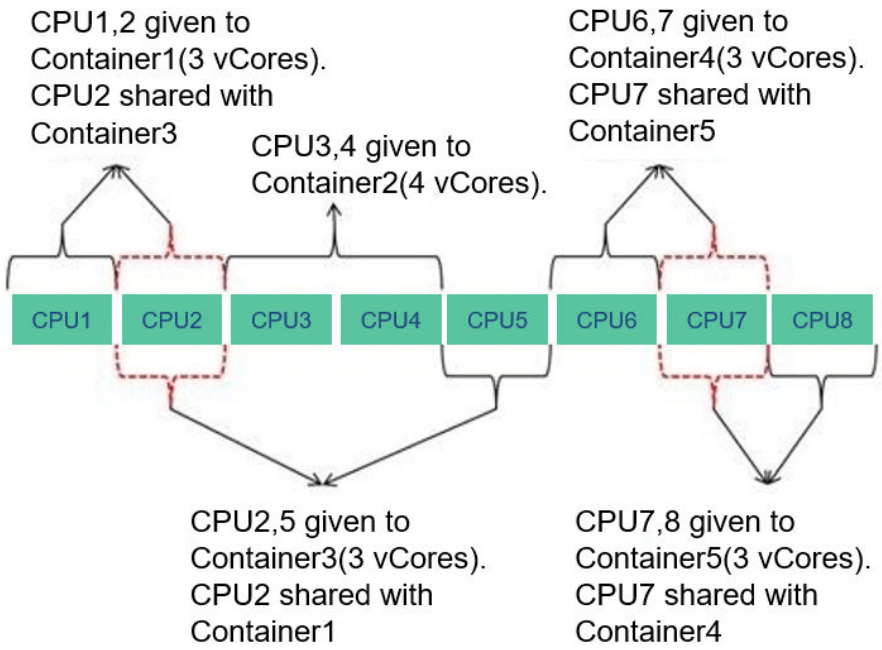
Aislamiento duro de CPU

Yarn no puede controlar estrictamente los recursos de CPU utilizados por cada container. Cuando se usa el subsistema de CPU, un container puede ocupar recursos excesivos. Por lo tanto, CPUset se utiliza para controlar la asignación de recursos.

Para resolver este problema, los recursos de CPU se asignan a cada contenedor basándose en la relación de núcleos virtuales (vCores) a núcleos físicos. Si un container requiere un núcleo

físico completo, el container lo tiene. Si un container necesita solamente algunos núcleos físicos, varios recipientes pueden compartir el mismo núcleo físico. La siguiente figura muestra un ejemplo de la cuota de CPU. La proporción dada de vCores a núcleos físicos es 2:1.

Figura 6-136 Cuota de CPU

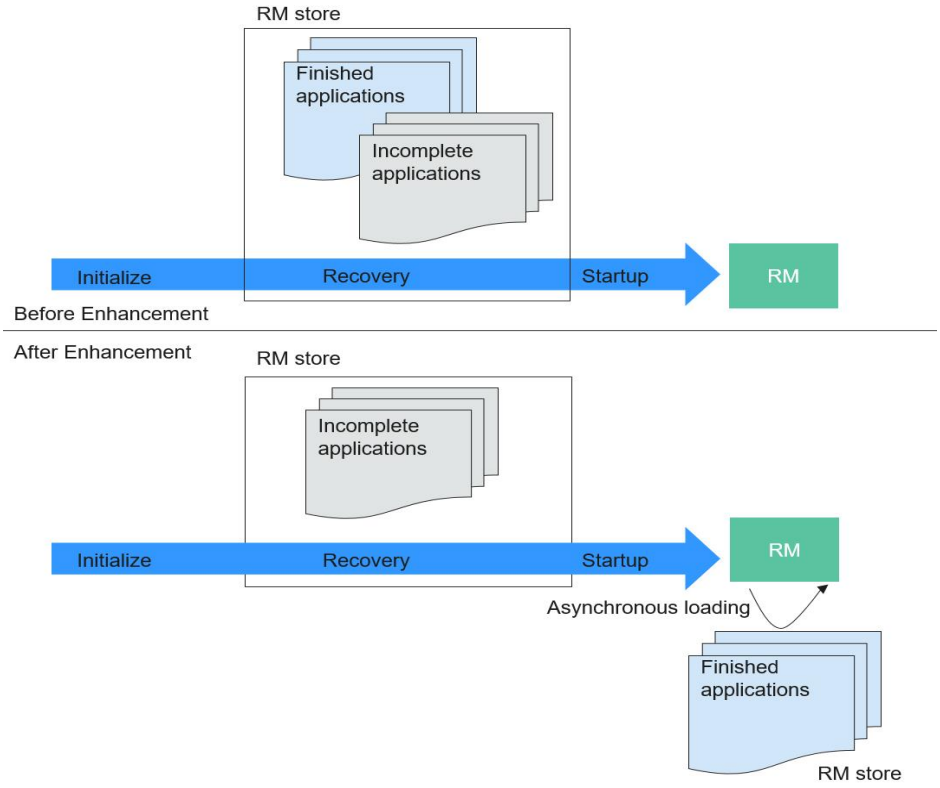


Función de código abierto mejorada: Optimización del rendimiento de reinicio

En general, el ResourceManager recuperado puede obtener aplicaciones en ejecución y completadas. Sin embargo, un gran número de aplicaciones completadas puede causar problemas tales como un arranque lento y un largo tiempo de ResourceManagers de conmutación/reinicio de HA.

Para acelerar el inicio, obtenga la lista de aplicaciones sin terminar antes de iniciar ResourceManagers. En este caso, la aplicación completada continúa siendo recuperada en el subproceso asíncrono de fondo. La siguiente figura muestra cómo se inicia la recuperación de ResourceManager.

Figura 6-137 Inicio de la recuperación de ResourceManager



6.32 ZooKeeper

6.32.1 Principios básicos de ZooKeeper

Descripción

ZooKeeper es un servicio de coordinación distribuido y de alta disponibilidad. ZooKeeper se utiliza para proporcionar las siguientes funciones:

- Evita que el sistema reciba SPOFs y proporciona servicios confiables para las aplicaciones.
- Proporciona servicios de coordinación distribuidos y gestiona la información de configuración.

Arquitectura

Los nodos de un clúster de ZooKeeper tienen tres funciones: Leader, Follower, y Observer, como se muestra en **Figura 6-138**. Generalmente, es necesario configurar un número impar de (2N+1) servicios de ZooKeeper en el clúster, y se requiere al menos (N+1) mayoría de votos para realizar con éxito la operación de escritura.

Figura 6-138 Arquitectura

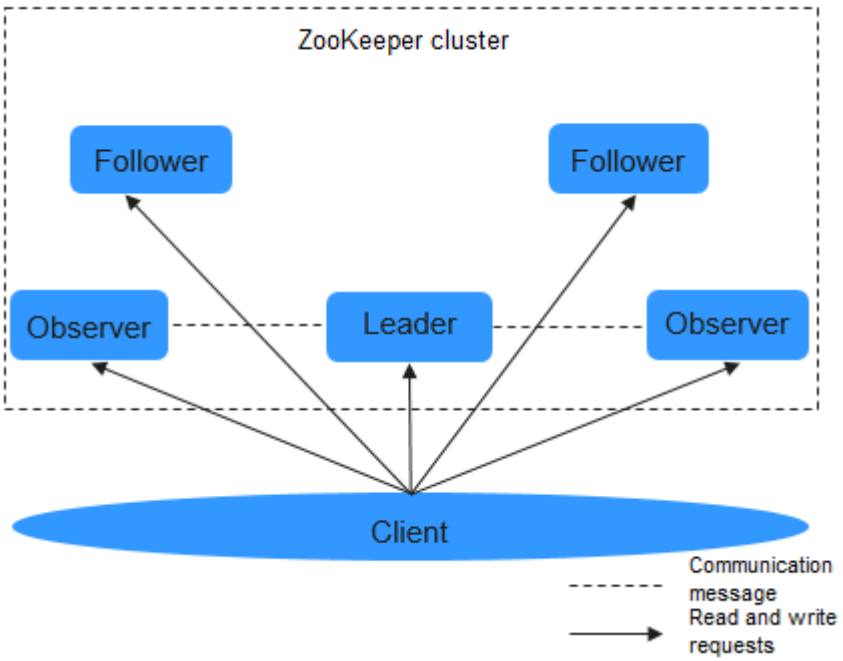


Tabla 6-31 describe las funciones de cada módulo mostrado en **Figura 6-138**.

Tabla 6-31 Descripción de la arquitectura

Nombre	Descripción
Leader	Solo un nodo sirve como líder en un clúster de ZooKeeper. El Leader, elegido por los Followers utilizando el protocolo de ZooKeeper Atomic Broadcast (ZAB), recibe y coordina todas las solicitudes de escritura y sincroniza la información escrita con los Followers y Observers.
Follower	Follower tiene dos funciones: ● Previene los SPOF. Un nuevo Leader es elegido de los Followers cuando el Leader es defectuoso. ● Procesa solicitudes de lectura e interactúa con el Leader para procesar solicitudes de escritura.
Observer	El Observer no participa en la votación para la elección ni escribe solicitudes. Solo procesa solicitudes de lectura y reenvía solicitudes de escritura al Leader, lo que aumenta la eficiencia del procesamiento del sistema.
Client	Lee y escribe datos desde o hacia el clúster ZooKeeper. Por ejemplo, HBase puede servir como un cliente ZooKeeper y utilizar la función de arbitraje del clúster de ZooKeeper para controlar el estado activo/en espera de HMaster.

Si los servicios de seguridad están habilitados en el clúster, se requiere autenticación durante la conexión a ZooKeeper. Los modos de autenticación son los siguientes:

- Modo de Keytab: necesita obtener un usuario humano-máquina del administrador del clúster de MRS para el inicio de sesión y autenticación de la consola de MRS, y obtener el archivo Keytab del usuario.
- Modo de ticket: obtenga un usuario humano-máquina del administrador del clúster MRS para el inicio de sesión seguro posterior, active las funciones renovables y reenviables del servicio Kerberos, establezca el período de actualización de ticket y reinicie Kerberos y componentes relacionados.

  NOTA

  - Por defecto, el período de validez de la contraseña de usuario es de 90 días. Por lo tanto, el período de validez del archivo Keytab obtenido es de 90 días.
  - Los parámetros para habilitar las funciones renovables y reenviables y establecer el intervalo de actualización del ticket se encuentran en la pestaña **System** de la página de configuración del servicio Kerberos. El intervalo de actualización del ticket se puede establecer en **kdc_renew_lifetime** o **kdc_max_renewable_life** en función de la situación real.

Principios

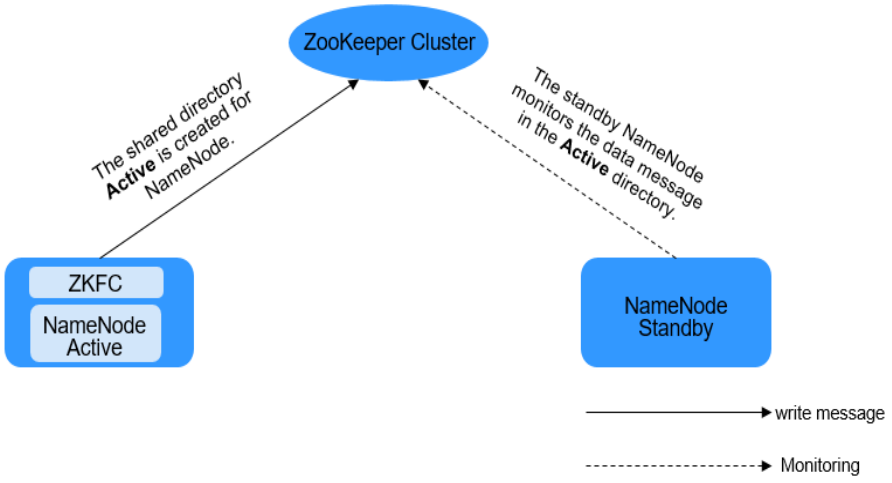
- **Solicitudes de escritura**
  a. Después de que el Follower o Observer recibe una solicitud de escritura, el Follower o Observer envía la solicitud al Leader.
  b. El Leader coordina a los Followers para determinar si acepta la solicitud de escritura mediante votación.
  c. Si más de la mitad de los votantes devuelven un mensaje de éxito de escritura, el Leader envía la solicitud de escritura y devuelve un mensaje de éxito. De lo contrario, se devuelve un mensaje de error.
  d. El Follower o Observer devuelve los resultados del procesamiento.
- **Solicitudes de solo lectura**
  El cliente lee directamente los datos del Leader, Follower, o Observer.

6.32.2 Relación entre ZooKeeper y otros componentes

Relación entre ZooKeeper y HDFS

Figura 6-139 muestra la relación entre ZooKeeper y HDFS.

Figura 6-139 Relación entre ZooKeeper y HDFS



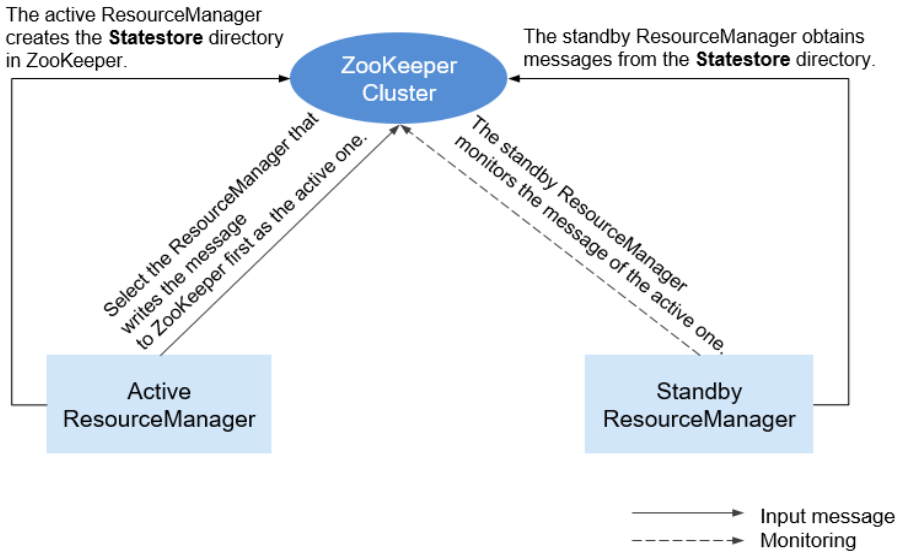
Como cliente de un clúster de ZooKeeper, ZKFailoverController (ZKFC) supervisa el estado de NameNode. ZKFC solo se implementa en el nodo donde reside NameNode y en el NameNodes HDFS activo y en espera.

1. El ZKFC se conecta a ZooKeeper y guarda información como nombres de host en ZooKeeper bajo el directorio de znode **/hadoop-ha**. NameNode que crea el directorio primero se considera como el nodo activo, y el otro es el nodo en espera. NameNodes lee la información de NameNode periódicamente a través de ZooKeeper.
2. Cuando el proceso del nodo activo finaliza de forma anormal, NameNode en espera detecta cambios en el directorio **/hadoop-ha** a través de ZooKeeper y, a continuación, se hace cargo del servicio del NameNode activo.

Relación entre ZooKeeper y YARN

Figura 6-140 muestra la relación entre ZooKeeper y YARN.

Figura 6-140 Relación entre ZooKeeper y YARN

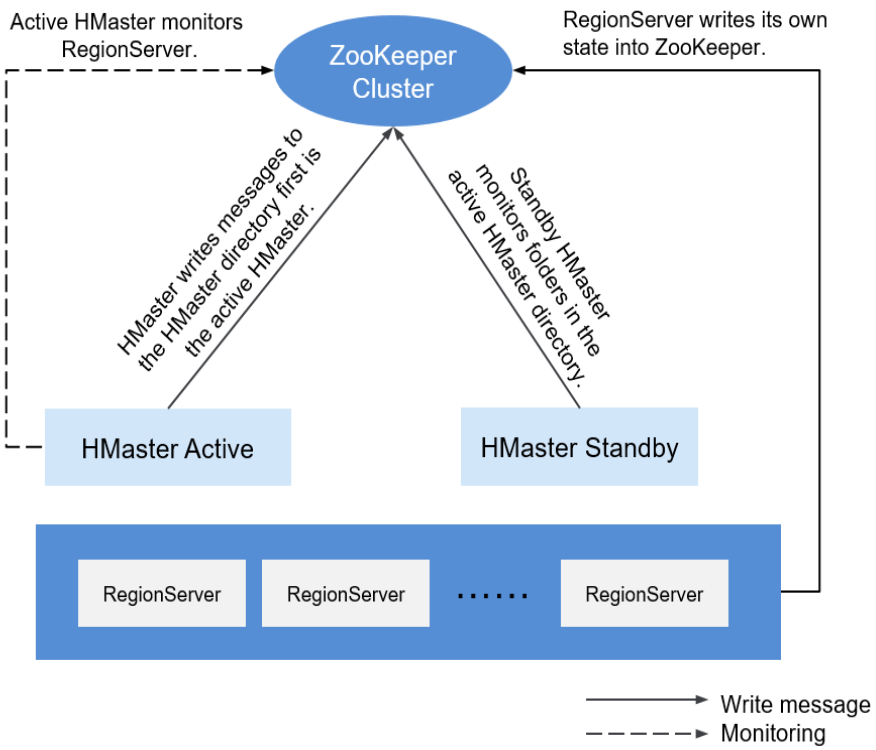


1. Cuando se inicia el sistema, ResourceManager intenta escribir información de estado en el ZooKeeper. ResourceManager que escribe primero información de estado en ZooKeeper se selecciona como ResourceManager activo, y otros son ResourceManagers en espera. Los ResourceManagers en espera monitorean periódicamente la información de elección activa de ResourceManager en ZooKeeper.
2. El ResourceManager activo crea el directorio **Statestore** de ZooKeeper para almacenar la información de la aplicación. Si el ResourceManager activo es defectuoso, el ResourceManager en espera obtiene la información de la aplicación del directorio **Statestore** y restaura los datos.

Relación entre ZooKeeper y HBase

Figura 6-141 muestra la relación entre ZooKeeper y HBase.

Figura 6-141 Relación entre ZooKeeper y HBase

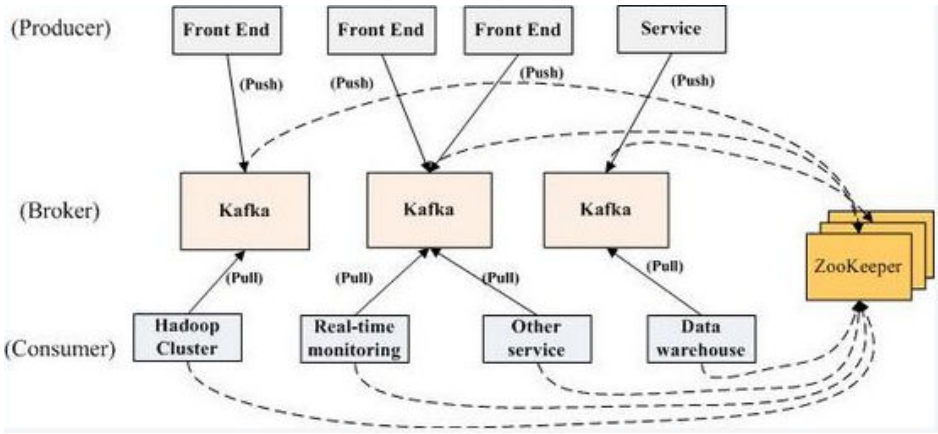


1. HRegionServer se registra en ZooKeeper en el nodo Efímero. ZooKeeper almacena la información de HBase, incluidos los metadatos de HBase y las direcciones HMaster.
2. HMaster detecta el estado de salud de cada HRegionServer usando ZooKeeper y los monitorea.
3. HBase admite varios nodos HMaster (como NameNodes HDFS). Cuando el HMatser activo está defectuoso, el HMaster en espera obtiene la información de estado de todo el clúster mediante ZooKeeper. Es decir, el uso de ZooKeeper puede evitar HBase SPOFs.

Relación entre ZooKeeper y Kafka

Figura 6-142 muestra la relación entre ZooKeeper y Kafka.

Figura 6-142 Relación entre ZooKeeper y Kafka



1. Broker utiliza ZooKeeper para registrar la información de broker y elegir partition leader.
2. El consumidor usa ZooKeeper para registrar información del consumidor, incluida la lista de particiones del consumidor. Además, se utiliza ZooKeeper para descubrir la lista de broker, establecer una conexión de socket con partition leader y obtener mensajes.

6.32.3 Funciones mejoradas de código abierto de ZooKeeper

Registro mejorado

En el modo de seguridad, se elimina un nodo efímero siempre que expire la sesión que creó el nodo. La eliminación de nodo efímero se registra en los registros de auditoría de modo que se puede obtener un estado de nodo efímero.

Los nombres de usuario deben agregarse a los registros de auditoría para todas las operaciones realizadas en los clientes de ZooKeeper.

En el cliente de ZooKeeper, cree un znode, cuyo principal de Kerberos es **zkcli/hadoop.**<*System domain name*>**@**<*System domain name*>.

Por ejemplo, abra el archivo **<ZOO_LOG_DIR>/zookeeper_audit.log**. El contenido del archivo es el siguiente:

```
2016-12-28 14:17:10,505 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test1?
result=success
2016-12-28 14:17:10,530 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test2?
result=success
2016-12-28 14:17:10,550 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test3?
result=success
2016-12-28 14:17:10,570 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test4?
result=success
2016-12-28 14:17:10,592 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test5?
result=success
2016-12-28 14:17:10,613 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test6?
result=success
2016-12-28 14:17:10,633 | INFO  | CommitProcWorkThread-4 |
session=0x12000007553b4903?user=10.177.223.78,zkcli/hadoop.hadoop.com@HADOOP.COM?
ip=10.177.223.78?operation=create znode?target=ZooKeeperServer?znode=/test7?
result=success
```

El contenido muestra que los registros del usuario cliente ZooKeeper **zkcli/hadoop.hadoop.com@HADOOP.COM** se agregan al registro de auditoría.

Detalles de usuario de ZooKeeper

En ZooKeeper, diferentes esquemas de autenticación utilizan diferentes credenciales como usuarios. En función del requisito del proveedor de autenticación, cualquier parámetro puede considerarse como usuarios.

Ejemplo:

- **SAMLAuthenticationProvider** utiliza el principal de cliente como usuario.
- **X509AuthenticationProvider** utiliza el certificado de cliente de usuario como usuario.
- **IAuthenticationProvider** utiliza la dirección IP del cliente como usuario.
- Se puede obtener un nombre de usuario del proveedor de autenticación personalizado implementando el método **org.apache.zookeeper.server.auth.ExtAuthenticationProvider.getUserName(String)**. Si el método no está implementado, se omitirá la obtención del nombre de usuario de la instancia del proveedor de autenticación.

Función de código abierto mejorada: comunicación de SSL de ZooKeeper (conexión de Netty)

El diseño ZooKeeper contiene el paquete Nio y no es compatible con SSL posterior a la versión 3.5. Para resolver este problema, se agrega Netty a ZooKeeper. Por lo tanto, si necesita usar SSL, habilite Netty y establezca los siguientes parámetros en el servidor y el cliente:

El servidor de código abierto solo admite contraseñas de texto sin formato, lo que puede causar problemas de seguridad. Por lo tanto, dichas contraseñas de texto ya no se utilizan en el servidor.

- Client
  a. Establezca **-Dzookeeper.client.secure** en el archivo **zkCli.sh/zkEnv.sh** en **true** para utilizar la comunicación segura en el cliente. A continuación, el cliente puede conectarse al secureClientPort en el servidor.
  b. Establezca los siguientes parámetros en el archivo **zkCli.sh/zkEnv.sh** para configurar el entorno del cliente:

Parámetro	Descripción
-Dzookeeper.clientCnxnSocket	Utilizado para la comunicación de Netty entre clientes. Valor predeterminado: **org.apache.zookeeper.ClientCnxnSocketNetty**
-Dzookeeper.ssl.keyStore.location	Indica la ruta de acceso para almacenar el archivo de keystore.
-Dzookeeper.ssl.keyStore.password	Cifra una contraseña.
-Dzookeeper.ssl.trustStore.location	Indica la ruta de acceso para almacenar el archivo Truststore.
-Dzookeeper.ssl.trustStore.password	Cifra una contraseña.
-Dzookeeper.config.crypt.class	Descifra una contraseña cifrada.
-Dzookeeper.ssl.password.encrypted	Valor predeterminado: **false** Si las contraseñas del almacén de claves y del almacén de confianza están cifradas, establezca este parámetro en **true**.

Parámetro	Descripción
-Dzookeeper.ssl.enabled.protocols	Define los protocolos de SSL que se habilitarán para el contexto SSL.
-Dzookeeper.ssl.exclude.cipher.ext	Define la lista de contraseñas separadas por una coma que debe excluirse del contexto SSL.

NOTA

Los parámetros anteriores se deben establecer en el archivo **zkCli.sh/zk.Env.sh**.

- Server
  a. Establezca **secureClientPort** en **3381** en el archivo **zoo.cfg**.
  b. Establezca **zookeeper.serverCnxnFactory** en **org.apache.zookeeper.server.NettyServerCnxnFactory** en el archivo **zoo.cfg** del servidor.
  c. Establezca los siguientes parámetros en el archivo **zoo.cfg** (en la ruta **zookeeper/conf/zoo.cfg**) para configurar el entorno del servidor:

Parámetro	Descripción
ssl.keyStore.location	Ruta de acceso para almacenar el archivo **keystore.jks**
ssl.keyStore.password	Cifra una contraseña.
ssl.trustStore.location	Indica la ruta de acceso para almacenar el archivo Truststore.
ssl.trustStore.password	Cifra una contraseña.
config.crypt.class	Descifra una contraseña cifrada.
ssl.keyStore.password.encrypted	Valor predeterminado: **false** Si este parámetro se establece en **true**, se puede usar la contraseña cifrada.
ssl.trustStore.password.encrypted	Valor predeterminado: **false** Si este parámetro se establece en **true**, se puede usar la contraseña cifrada.
ssl.enabled.protocols	Define los protocolos de SSL que se habilitarán para el contexto SSL.
ssl.exclude.cipher.ext	Define la lista de contraseñas separadas por una coma que debe excluirse del contexto SSL.

  d. Inicie ZKserver y conecte el cliente de seguridad al puerto de seguridad.
- Credencial

La credencial utilizada entre el cliente y el servidor de ZooKeeper es **X509AuthenticationProvider**. Esta credencial se inicializa con los certificados de servidor especificados y de confianza mediante los siguientes parámetros:

- zookeeper.ssl.keyStore.location
- zookeeper.ssl.keyStore.password
- zookeeper.ssl.trustStore.location
- zookeeper.ssl.trustStore.password

NOTA

Si no desea utilizar el mecanismo predeterminado de ZooKeeper, puede configurarse con diferentes mecanismos de confianza según sea necesario.

7 Funciones

7.1 Multitenant

Introducción a las funciones

Los clústeres de datos de las empresas modernas se están desarrollando hacia la centralización y la cloudificación. Los clústeres de big data de clase empresarial deben cumplir los siguientes requisitos:

- Llevar datos de diferentes tipos y formatos y ejecutar trabajos y aplicaciones de diferentes tipos (análisis, consultas y procesamiento de secuencias).
- Aislar los datos de un usuario de los de otro usuario que tiene requisitos exigentes en materia de seguridad de datos, como un banco o un instituto gubernamental.

Los requisitos anteriores traen los siguientes desafíos para el clúster de big data:

- Asignación y programación adecuadas de recursos para garantizar un funcionamiento estable de aplicaciones y trabajos
- Control de acceso estricto para garantizar la seguridad de los datos y el servicio

Multitenant aísla los recursos de un clúster de big data en conjuntos de recursos. Los usuarios pueden arrendar los conjuntos de recursos deseados para ejecutar aplicaciones y trabajos y almacenar datos. En un clúster de big data, se pueden implementar varios conjuntos de recursos para satisfacer diversos requisitos de varios usuarios.

El clúster de big data de MRS proporciona una solución completa de big data de multitenant de clase empresarial. Multitenant es una colección de múltiples recursos (cada conjunto de recursos es un tenant) en un clúster de big data de MRS. Puede asignar y programar recursos, incluidos los recursos informáticos y de almacenamiento.

Ventajas

- Configuración y aislamiento de recursos adecuados

  Los recursos de un tenant están aislados de los de otro tenant. El uso de recursos de un tenant no afecta a otros tenants. Este mecanismo garantiza que cada tenant pueda configurar los recursos en función de los requisitos de servicio, lo que mejora la utilización de los recursos.

- Medición y estadísticas del consumo de recursos

  Los tenants son solicitantes de recursos del sistema y consumidores. Los recursos del sistema se planifican y asignan en función de los tenants. El consumo de recursos por parte de los tenants se puede medir y registrar.

- Seguridad de datos y seguridad de acceso garantizada

  En escenarios de multitenant, los datos de cada inquilino se almacenan por separado para garantizar la seguridad de los datos. El acceso a los recursos de tenants se controla para garantizar la seguridad del acceso.

Programadores mejorados

Los planificadores se dividen en el programador de capacidad de código abierto y el programador Superior propietario de Huawei.

Para cumplir con los requisitos de la empresa y hacer frente a los desafíos que enfrenta la comunidad Yarn en la programación, Huawei desarrolla el programador Superior. Además de heredar las ventajas del programador de Capacity y del programador de Fair, este programador se mejora en los siguientes aspectos:

- Política mejorada de uso compartido de recursos

  El programador Superior admite la jerarquía de colas. Integra las funciones de programadores de código abierto y comparte recursos basados en políticas configurables. En términos de instancias, los administradores del clúster de MRS pueden usar el programador Superior para configurar un valor absoluto o una política de porcentaje para los recursos de cola. La política de uso compartido de recursos del programador Superior mejora la política de programación de etiquetas de Yarn como una característica de fondo de recursos. Los nodos del clúster de Yarn se pueden agrupar en función de la capacidad o el tipo de servicio para garantizar que las colas puedan utilizar los recursos de manera más eficiente.

- Política de reserva de recursos basada en tenant

  Los recursos requeridos por los inquilinos deben estar asegurados para ejecutar tareas críticas. El programador Superior construye un mecanismo para soportar la política de reserva de recursos. Al hacerlo, los recursos reservados se pueden asignar a las tareas ejecutadas por las colas de inquilinos de una manera oportuna para garantizar la correcta ejecución de la tarea.

- Compartición justa entre tenants y usuarios de grupo de recurso

  El programador Superior permite configurar recursos compartidos para los usuarios en una cola. Cada tenant puede tener usuarios con diferentes ponderaciones. Los usuarios muy ponderados pueden requerir más recursos compartidos.

- Rendimiento de programación garantizado en un gran clúster

  El programador Superior recibe los latidos de cada NodeManager y guarda la información de recursos en la memoria, lo que permite al programador controlar el uso de recursos del clúster globalmente. El planificador Superior utiliza el modelo de programación push, lo que hace que la programación sea más precisa y eficiente y mejora notablemente la utilización de los recursos del clúster. Además, el programador Superior ofrece un excelente rendimiento cuando el intervalo entre los latidos NodeManager es largo y evita las tormentas de latidos en grupos grandes.

- Política de prioridad

  Si no se puede cumplir el requisito mínimo de recursos de un servicio después de que el servicio obtenga todos los recursos disponibles, se produce una preferencia. La función de preferencia está deshabilitada por defecto.

7.2 Mejoras de seguridad

MRS es una plataforma para la gestión y el análisis de datos masivos y tiene alta seguridad. MRS protege los datos del usuario y el servicio que se ejecuta de los siguientes aspectos:

- Aislamiento de red

  Todo el sistema se implementa en una VPC en la nube pública para proporcionar un entorno de red aislado y garantizar la seguridad de servicio y gestión del clúster. Al combinar las funciones de división de subred, control de ruta y grupo de seguridad de VPC, MRS proporciona un entorno de red aislado seguro y confiable.

- Aislamiento de recursos

  MRS admite la implementación de recursos y el aislamiento de recursos físicos en zonas dedicadas. Puede combinar de forma flexible recursos informáticos y de almacenamiento, como recursos informáticos dedicados + recursos de almacenamiento compartidos, recursos informáticos compartidos + recursos de almacenamiento dedicados y recursos informáticos dedicados + recursos de almacenamiento dedicados.

- Seguridad del host

  MRS se puede integrar con servicios de seguridad en la nube pública, incluidos Vulnerability Scan Service (VSS), Host Security Service (HSS), Web Application Firewall (WAF), Cloud Bastion Host (CBH) y Web Tamper Protection (WTP). Las siguientes medidas se proporcionan por Huawei Cloud para mejorar la seguridad del sistema operativo y los puertos:

  - Mejoras de la seguridad de kernels del sistema operativo
  - Control de permisos del sistema operativo
  - Gestión de puertos del sistema operativo

- Seguridad de las aplicaciones

  Se utilizan las siguientes medidas para garantizar el funcionamiento normal de los servicios de big data:

  - Identificación y autenticación
  - Seguridad de las aplicaciones de web
  - Control de acceso
  - Seguridad de auditoría
  - Seguridad de contraseñas

- Seguridad de datos

  Se proporcionan las siguientes medidas para garantizar la confidencialidad, integridad y disponibilidad de cantidades masivas de datos de usuario:

  - Recuperación ante desastres: MRS soporta respaldo de datos a OBS y alta confiabilidad entre regiones.
  - Copia de respaldo: MRS admite la copia de respaldo de metadatos de DBService, LDAP y NameNode y la copia de respaldo de datos de servicio HDFS y HBase.

- Integridad de los datos

  Los datos se verifican para garantizar su integridad durante el almacenamiento y la transmisión.

  - CRC32C se utiliza por defecto para verificar la exactitud de los datos de usuario almacenados en HDFS.

  - DataNodes de HDFS almacenan los datos verificados. Si los datos transmitidos desde un cliente son anormales (incompletos), los DataNodes informan de la anormalidad al cliente y el cliente reescribe los datos.
  - El cliente comprueba la integridad de los datos cuando lee datos de un DataNode. Si los datos están incompletos, el cliente leerá los datos de otro DataNode.

- confidencialidad de los datos

  Basado en Apache Hadoop, el sistema de archivos distribuido de MRS admite el almacenamiento cifrado de archivos para evitar que los datos confidenciales se almacenen en texto plano, mejorando la seguridad de los datos. Las aplicaciones solo necesitan cifrar los datos confidenciales especificados. Los servicios no se ven afectados durante el proceso de cifrado. Basado en el cifrado de datos del sistema de archivos, Hive proporciona cifrado a nivel de tabla y HBase proporciona cifrado a nivel de familia de columnas. Los datos confidenciales se pueden cifrar y almacenar después de especificar un algoritmo de encriptación durante la creación de la tabla.

  El almacenamiento cifrado y el control de acceso de los datos se utilizan para garantizar la seguridad de los datos del usuario.

  - HBase almacena los datos de servicio en el HDFS después de la compresión. Los usuarios pueden configurar el algoritmo de cifrado de AES y SMS4 para cifrar datos.
  - Todos los componentes permiten establecer permisos de acceso para directorios de datos locales. Los usuarios no autorizados no pueden acceder a los datos.
  - Toda la información del usuario del clúster se almacena en texto cifrado.

- Autenticación de seguridad
  - Utiliza un sistema de autenticación unificado de usuario y basado en roles, así como un modelo de control de acceso basado en cuentas y roles (RBAC) para controlar de forma centralizada los permisos de usuario y gestionar por lotes la autorización de usuario.
  - Utiliza el protocolo ligero de acceso a directorios (LDAP) como sistema de gestión de cuentas y realiza la autenticación de Kerberos en las cuentas.
  - Proporciona la función de inicio de sesión único (SSO) que gestiona y autentica de forma centralizada a los usuarios de componentes y sistemas de MRS.
  - Audita a los usuarios que han iniciado sesión en Manager.

7.3 Fácil acceso a Web UIs de componentes

Los componentes de Big Data tienen sus propias interfaces de usuario web para gestionar sus propios sistemas. Sin embargo, no puede acceder fácilmente a las interfaces de usuario web debido al aislamiento de la red. Por ejemplo, para acceder a la interfaz de usuario web HDFS, debe crear un ECS para iniciar sesión de forma remota en la interfaz de usuario web. Esto hace que el acceso a la interfaz de usuario sea complejo y poco amigable.

MRS proporciona un canal seguro basado en EIP para que pueda acceder fácilmente a las interfaces de usuario web de los componentes. Esto es más conveniente que vincular una EIP por sí mismo, y puede acceder a las interfaces de usuario web con unos pocos clics, evitando los pasos para iniciar sesión en una VPC, agregando reglas de grupo de seguridad y obteniendo una dirección IP pública. Para los componentes Hadoop, Spark, HBase y Hue en los clústeres de análisis y el componente Storm en los clústeres de streaming, puede acceder rápidamente a sus interfaces de usuario web desde las entradas en Manager.In addition, an EIP is used for access in a unified manner, which is easy to use and remember.

7.4 Mejora de la confiabilidad

Basado en el software de código abierto Apache Hadoop, MRS optimiza y mejora la confiabilidad y el rendimiento de los principales componentes del servicio.

Confiabilidad del sistema

- HA para todos los nodos de gestión

  En la versión de código abierto de Hadoop, los datos y los nodos de cómputo se gestionan en un sistema distribuido, en el que un único punto de fallo (SPOF) no afecta al funcionamiento de todo el sistema. Sin embargo, un SPOF puede ocurrir en nodos de gestión que se ejecutan en modo centralizado, lo que se convierte en la debilidad de la confiabilidad global del sistema.

  MRS proporciona mecanismos de doble nodo similares para todos los nodos de gestión de los componentes de servicio, como Manager, HDFS NameNodes, HiveServers, HBase HMasters, Yarn ResourceManagers, KerberosServers, y LdapServers. Todos ellos se implementan en modo activo/en espera o se configuran con carga compartida, lo que evita que los SPOF afecten la confiabilidad del sistema.

- Garantía de confiabilidad en caso de excepciones.

  Mediante el análisis de confiabilidad, se proporcionan las siguientes medidas para manejar excepciones de software y hardware para mejorar la confiabilidad del sistema:

  - Después de restaurar la fuente de alimentación, los servicios se ejecutan correctamente independientemente de un fallo de alimentación de un solo nodo o de todo el clúster, lo que garantiza la confiabilidad de los datos en caso de fallos de alimentación inesperados. Los datos clave no se perderán a menos que el disco duro esté dañado.
  - Las comprobaciones del estado de salud y el manejo de fallos del disco duro no afectan a los servicios.
  - Las fallas del sistema de archivos se pueden manejar automáticamente y los servicios afectados se pueden restaurar automáticamente.
  - Las fallas del proceso y del nodo se pueden manejar automáticamente, y los servicios afectados se pueden restaurar automáticamente.
  - Las fallas de la red se pueden manejar automáticamente y los servicios afectados se pueden restaurar automáticamente.

- Copia de resapldo y restauración de datos

  MRS proporciona funciones de copia de respaldo completa, copia de respaldo incremental y restauración basadas en los requisitos de servicio, evitando el impacto de la pérdida de datos y daños en los servicios y asegurando una rápida restauración del sistema en caso de excepciones.

  - Copia de respaldo automática

    MRS proporciona una copia de respaldo automática de los datos en Manager. Según la política de copia de respaldo personalizada, los datos de los clústeres, incluidos los datos de LdapServer y DBService, se pueden hacer copias de respaldo automáticamente.

  - Copia de respaldo manual

También puede realizar una copia de seguridad manual de los datos del sistema de gestión de clústeres antes de ampliación de capacidad, y la instalación de parches para recuperar las funciones del sistema de gestión de clústeres en caso de fallas.

Para mejorar la confiabilidad del sistema, los datos en Manager y HBase se respaldan manualmente en un servidor de terceros.

confiabilidad del nodo

- Supervisión del estado del sistema operativo

  MRS recopila periódicamente datos de uso de recursos de hardware del sistema operativo, incluido el uso de CPU, memoria, discos duros y recursos de red.

- Supervisión del estado del proceso

  MRS comprueba el estado de las instancias de servicio y los indicadores de estado de los procesos de instancia de servicio, lo que le permite conocer el estado de estado de los procesos de manera oportuna.

- Solución automática de problemas de disco

  MRS está mejorado basado en la versión de código abierto. Puede supervisar el estado del hardware y los sistemas de archivos en todos los nodos. Si se produce una excepción, las particiones correspondientes se eliminarán del grupo de almacenamiento. Si un disco está defectuoso y se reemplaza, se agregará un nuevo disco duro para ejecutar los servicios. En este caso, se simplifican las operaciones de mantenimiento. La sustitución de los discos defectuosos se puede completar en línea. Además, los usuarios pueden configurar discos de copia de respaldo en caliente para reducir el tiempo de restauración del disco defectuoso y mejorar la confiabilidad del sistema.

- Configuración de LVM para discos de nodo

  MRS le permite configurar Logic Volume Management (LVM) para planificar varios discos como un grupo de volúmenes lógicos. La configuración de LVM puede evitar el uso desigual de los discos. Es especialmente importante garantizar el uso uniforme de discos en componentes que pueden usar múltiples capacidades de disco, como HDFS y Kafka. Además, LVM admite la expansión de la capacidad del disco sin volver a conectarlo, lo que evita la interrupción del servicio.

Confiabilidad de los datos

MRS puede usar los grupos de nodos de antiafinidad y las capacidades de grupos de colocación proporcionadas por ECS y la capacidad de reconocimiento de rack de Hadoop para distribuir datos de manera redundante a múltiples máquinas host físicas, evitando la pérdida de datos causada por fallas de hardware físico.

7.5 Gestión de trabajos

La función de gestión de trabajos proporciona una entrada para que envíe trabajos en un clúster, incluidos los trabajos MapReduce, Spark, HiveQL y SparkSQL. MRS trabaja con el estudio DataArts de Huawei Cloud para proporcionar un entorno de desarrollo de colaboración de big data único y capacidades de programación de big data totalmente gestionadas, lo que le ayuda a construir centros de procesamiento de big data sin esfuerzo.

DataArts le permite desarrollar y depurar scripts MRS HiveQL/SparkSQL en línea y desarrollar trabajos MRS mediante la realización de operaciones de arrastrar y soltar para migrar e integrar datos entre MRS y más de 20 fuentes de datos heterogéneas. La potente

programación de trabajos y el monitoreo flexible y las alarmas le ayudan a gestionar fácilmente los datos y O&M de trabajo.

7.6 Acciones de arranque

Introducción a las funciones

MRS proporciona clústeres de big data elásticos estándar en la nube. Se pueden instalar e implementar nueve componentes de big data, como Hadoop y Spark. Actualmente, los clústeres de big data en la nube estándar no pueden cumplir con todos los requisitos del usuario, por ejemplo, en los siguientes escenarios:

- Las configuraciones comunes del sistema operativo no pueden cumplir con los requisitos de procesamiento de datos, por ejemplo, aumentar el número máximo de conexiones del sistema.
- Es necesario instalar herramientas de software o entornos de ejecución, por ejemplo, Gradle y el paquete de lenguaje R de dependencia.
- Los paquetes de componentes de Big Data deben modificarse en función de los requisitos de servicio, por ejemplo, modificando el paquete de instalación de Hadoop o Spark.
- Es necesario instalar otros componentes de big data que no son compatibles con MRS.

Para cumplir los requisitos de personalización anteriores, puede realizar operaciones manualmente en los nodos existentes y los nuevos agregados. El proceso general es complejo y propenso a errores. Además, las operaciones manuales no se pueden rastrear y los datos no se pueden procesar inmediatamente después de crear un clúster basado en su demanda.

Por lo tanto, MRS admite acciones de arranque personalizadas que permiten ejecutar secuencias de comandos en un nodo especificado antes o después de iniciar un componente de clúster. Puede ejecutar acciones de arranque para instalar software de terceros que no sea compatible con MRS, modificar el entorno de ejecución del clúster y realizar otras personalizaciones. Si elige ejecutar acciones de arranque al expandir un clúster, las acciones de arranque se ejecutarán en los nodos recién agregados de la misma manera. MRS ejecuta el script especificado como **root** de usuario. Puede ejecutar el comando de **su - *xxx*** en el script para cambiar al usuario.

Beneficios para el cliente

Puede utilizar las acciones de arranque personalizadas para configurar de forma flexible y sencilla sus clústeres dedicados y personalizar la instalación de software.

7.7 Gestión de proyecto empresarial

Un proyecto empresarial es un modo de gestión de recursos en la nube. Gestión empresarial proporciona a los usuarios una gestión integral de recursos basados en la nube, personal, permisos y finanzas. Las consolas de gestión comunes están orientadas al control y configuración de productos en la nube individual. Por el contrario, la consola de Enterprise Management está más centrada en la gestión de recursos. Está diseñado para ayudar a las empresas a gestionar recursos, personal, permisos y finanzas basados en la nube, de manera jerárquica, como la gestión de empresas, departamentos y proyectos.

MRS permite a los usuarios que han habilitado Enterprise Project Management Service (EPS) configurar proyectos empresariales para un clúster durante la creación de clústeres y utilizar EPS para gestionar recursos MRS por grupo.

- Los usuarios pueden gestionar varios recursos por grupo.
- Los usuarios pueden ver información de recursos y detalles de gastos de proyectos de empresa.
- Los usuarios pueden controlar los permisos de acceso a nivel de proyecto de empresa.
- Los usuarios pueden consultar información financiera detallada por proyecto de empresa, incluidos pedidos, resumen de gastos y detalles de gastos.

7.8 Metadatos

MRS proporciona múltiples métodos de almacenamiento de metadatos. Al desplegar Hive y Ranger durante la creación del clúster de MRS, seleccione uno de los siguientes modos de almacenamiento según sea necesario:

- **Local**: Los metadatos se almacenan en la GaussDB local de un clúster. Cuando se elimina el clúster, también se eliminan los metadatos. Para conservar los metadatos, realice una copia de respaldo manual de los metadatos en la base de datos con antelación.
- **Conexión de datos externa**: Puede seleccionar **RDS PostgreSQL database** o **RDS MySQL database** asociado a la misma VPC y subred que el clúster actual. Los metadatos se almacenan en la base de datos y no se eliminan cuando se elimina el clúster actual. Varios clústeres de MRS pueden compartir los mismos metadatos.

NOTA

Hive en MRS 1.9.*x* o posterior permite especificar un método de almacenamiento de metadatos.

Ranger en MRS 1.9.*x* permite almacenar metadatos solo en la base de datos de MySQL asociada del servicio RDS.

7.9 Gestión de clústeres

7.9.1 Gestión del ciclo de vida de clústeres

MRS admite la gestión del ciclo de vida de clúster, incluida la creación y terminación de clústeres.

- Creación de un clúster: Después de especificar un tipo de clúster, componentes, número de nodos de cada tipo, especificaciones de VM, zona de disponibilidad, VPC e información de autenticación, MRS crea automáticamente un clúster que cumple con los requisitos de configuración. Puede ejecutar scripts personalizados en el clúster. Además, puede crear clústeres de diferentes tipos para múltiples escenarios de aplicaciones, como clústeres de análisis de Hadoop, clústeres de HBase y clústeres de Kafka. La plataforma de big data admite la implementación de clústeres heterogéneos. Es decir, las máquinas virtuales de diferentes especificaciones se pueden combinar en un clúster basado en los tipos de CPU, las capacidades de disco, los tipos de disco y los tamaños de memoria. Se pueden mezclar varias especificaciones de VM en un clúster.
- Terminación de un clúster: puede terminar un clúster de pago por uso que ya no sea necesario (incluidos los datos y las configuraciones del clúster). MRS eliminará todos los recursos relacionados con el clúster.

- Renovación: MRS ofrece dos modos de facturación: pago por uso y anual/mensual. En el modo de pago por uso, las tarifas se deducen cada hora y un saldo insuficiente puede conducir a pagos atrasados. En modo anual/mensual, los clústeres deben renovarse antes de que caduquen. Si su suscripción para el clúster de pago por uso o anual/mensual no se renueva, sus servicios seguirán ejecutándose, pero entrarán en un período de retención, durante el cual los clústeres MRS dejarán de ejecutarse pero se conservarán los datos.
- Cancelación de suscripción: si ha adquirido un clúster anual/mensual y no necesita los recursos del clúster antes de que caduquen los recursos del clúster, puede cancelar la suscripción a los recursos del clúster en MRS.

Compra de un clúster

En la consola de gestión de MRS, puede comprar un clúster MRS de pago por uso o anual/mensual. Puede seleccionar una región y especificaciones de recursos en la nube para comprar un clúster MRS que sea adecuado para servicios empresariales con un solo clic. MRS instala e implementa automáticamente la plataforma de big data de nivel empresarial de Huawei Cloud y optimiza los parámetros en función del tipo de clúster, la versión y las especificaciones de nodo seleccionados.

MRS le proporciona clústeres de big data totalmente gestionados. Al crear un clúster, puede establecer un modo de inicio de sesión de VM (contraseña o par de claves). Puede utilizar todos los recursos del clúster MRS creado. Además, MRS le permite implementar un clúster de big data en solo dos ECS con 4 vCPU y 8 GB de memoria, lo que proporciona opciones más flexibles para pruebas y desarrollo.

Los grupos MRS se clasifican en clústeres de análisis, streaming e híbridos.

- Clúster de análisis: Se utiliza para el análisis de datos fuera de línea y proporciona componentes de Hadoop.
- Clúster de streaming: Se utiliza para tareas de streaming y proporciona componentes de procesamiento de flujos.
- Clúster híbrido: se utiliza no solo para el análisis de datos sin conexión, sino también para el procesamiento de streaming, y proporciona componentes de Hadoop y componentes de procesamiento de streaming.
- Personalizado: puede combinar de forma flexible los componentes necesarios (MRS 3.x y versiones posteriores) según los requisitos de servicio.

Los nodos de clúster de MRS se clasifican en nodos de Master, Core, y Task.

- Nodo de Master: nodo de gestión en un clúster. Los procesos de Master de un sistema distribuido, Manager y bases de datos se despliega en los nodos de Master. Los nodos principales no pueden escalarse horizontalmente. La capacidad de procesamiento de los nodos principales determina el límite superior de la capacidad de gestión de todo el clúster. El MRS admite el escalamiento vertical de las especificaciones del nodo principal para proporcionar soporte para la gestión de un clúster más grande.
- Nodo de Core: se utiliza tanto para almacenamiento como para computación y se puede escalar horizontal o verticalmente. Dado que los nodos de Core soportan almacenamiento de datos, hay muchas restricciones sobre el escalado para evitar la pérdida de datos y no se puede realizar el escalado automático.
- Nodo de tarea: se utiliza solo para computación y se puede escalar horizontal y verticalmente. Los nodos de Task solo admiten tareas de cómputo. Por lo tanto, se puede realizar el escalamiento automático.

Puede comprar un clúster en dos modos: compra personalizada y compra rápida.

- Compra personalizada: en la página **Custom Config**, puede configurar de forma flexible los parámetros del clúster en función de los escenarios de la aplicación, como el modo de facturación y las especificaciones de ECS para adaptarse mejor a sus requisitos de servicio.
- Compra rápida: En la página **Quick Config**, puede comprar rápidamente un clúster basado en escenarios de aplicación, lo que mejora la eficiencia de la configuración del clúster. Actualmente, los clústeres de análisis Hadoop, los clústeres HBase y los clústeres Kafka están disponibles para su compra rápida.
  - Clúster de análisis de Hadoop: utiliza componentes en el ecosistema de Hadoop de código abierto para analizar y consultar grandes cantidades de datos. Por ejemplo, utilice Yarn para gestionar los recursos de clústeres; Hive y Spark para proporcionar almacenamiento fuera de línea y opciones de procesamiento de datos distribuidos a gran escala; Spark Streaming y Flink para ofrecer cómputo de datos de transmisiones; Presto para permitir consultas interactivas y Tez para proporcionar un marco de cómputo distribuido de grafos acíclicos dirigidos (DAG).
  - Clúster HBase: utiliza los componentes Hadoop y HBase para proporcionar un sistema de almacenamiento en la nube distribuido orientado a columnas que ofrece confiabilidad mejorada, rendimiento excelente y escalabilidad elástica. Es aplicable al almacenamiento y al cómputo distribuido de cantidades masivas de datos. HBase puede utilizarse para construir un sistema de almacenamiento capaz de almacenar datos a nivel de TB o incluso de PB. Con HBase, puede filtrar y analizar datos con facilidad, y obtener respuestas en milisegundos, así como extraer valor de los datos rápidamente.
  - Clúster de Kafka: utiliza Kafka y Storm para proporcionar un sistema de mensajes de código abierto con alto rendimiento y escalabilidad. Se utiliza ampliamente para recopilar registros y agregar datos de monitoreo con el fin de recabar datos de transmisión de forma eficiente, así como para el procesamiento y el almacenamiento de datos en tiempo real.

Terminación de un clúster

MRS le permite terminar un clúster cuando ya no es necesario. Una vez finalizado el clúster, se liberarán todos los recursos de nube utilizados por el clúster. Antes de terminar un clúster, se recomienda migrar o realizar copias de respaldo de los datos. Terminar el clúster solo cuando no se está ejecutando ningún servicio en el clúster o si el clúster es anormal y no puede proporcionar servicios basados en el análisis de O&M. Si los datos se almacenan en discos EVS o discos de paso a través en un clúster de big data, los datos se eliminarán una vez finalizado el clúster. Por lo tanto, tenga cuidado al terminar un clúster.

7.9.2 Escalamiento de clústeres

La capacidad de procesamiento de un clúster de big data se puede ampliar horizontalmente agregando nodos. Si la escala del clúster no cumple con los requisitos de servicio, puede escalar horizontal o verticalmente manualmente en el clúster. MRS selecciona inteligentemente el nodo con la menor carga o la cantidad mínima de datos a migrar para escalar verticalmente. El nodo que se va a escalar no recibirá nuevas tareas y continúa ejecutando las tareas existentes. Al mismo tiempo, MRS copia sus datos a otros nodos y se retira el nodo. Si las tareas en el nodo no se pueden completar después de mucho tiempo, MRS migra las tareas a otros nodos, minimizando el impacto en los servicios de clúster.

Escalamiento horizontal de un clúster

Actualmente, puede agregar nodos Core o Task para escalar un clúster para manejar cargas de servicio máximas. La adición de nodos de clúster MRS no afecta a los servicios del clúster existente. Para obtener más información sobre cómo rectificar el sesgo de datos causado por la expansión de la capacidad.

Escalar un clúster cargado en modo anual/mensual

Si la tasa de crecimiento del servicio supera el valor esperado después de suscribirse a un clúster MRS cargado en modo **Yearly/Monthly**, se requiere escalar el clúster más allá de su suscripción. MRS le permite escalar los clústeres cargados en modo **Yearly/Monthly** mientras disfruta de los descuentos de suscripción.

Puede acceder a la consola de gestión de MRS y agregar nodos a un clúster con unos pocos clics. El proceso de escalamiento horizontal del clúster no requiere intervención manual y toma solo unos minutos, lo que ayuda a aliviar la presión sobre las crecientes necesidades de procesamiento de datos de servicio.

Escalamiento en un clúster

Puede reducir el número de nodos Core o Task a escalar en un clúster para que MRS ofrezca mejores capacidades de almacenamiento e informática con menores costos de operación según los requisitos de servicio. Después de escalar en un clúster MRS, MRS selecciona automáticamente los nodos que se pueden escalar en función del tipo de servicios instalados en los nodos.

Durante el escalado de los nodos Core, se migran los datos de los nodos originales. Si la ubicación de datos se almacena en caché, el cliente actualiza automáticamente la información de ubicación, lo que puede afectar a la latencia. El escalado de nodo puede afectar a la duración de la respuesta del primer acceso a algunos datos de HBase en HDFS. Puede reiniciar HBase o deshabilitar o habilitar tablas relacionadas para evitar este problema.

Los nodos de tarea no almacenan datos del clúster. Son nodos de computación y no implican la migración de datos en los nodos.

7.9.3 Escalamiento automático

Introducción a las funciones

Cada vez más empresas utilizan tecnologías como Spark y Hive para analizar datos. El procesamiento de una gran cantidad de datos consume enormes recursos y cuesta mucho. Por lo general, las empresas analizan regularmente los datos en un período de tiempo fijo todos los días en lugar de todo el día. Para cumplir con los requisitos de las empresas, MRS proporciona la función de escalado automático para solicitar recursos adicionales durante las horas pico y liberar recursos durante las horas fuera de pico. Esto permite a los usuarios utilizar los recursos bajo demanda y centrarse en el negocio principal a costos más bajos.

En aplicaciones de big data, especialmente en escenarios de análisis y procesamiento de datos periódicos, los recursos informáticos de clúster deben ajustarse dinámicamente en función de los cambios en los datos de servicio para cumplir con los requisitos de servicio. La función de escalado automático de MRS permite que los clústeres sean escalados elásticamente o en función de las cargas de clúster. Además, si el volumen de datos cambia regularmente y desea escalar o en un clúster antes de que cambie el volumen de datos, puede utilizar la función de plan de recursos de MRS.

MRS admite dos tipos de políticas de escalado automático: reglas de escalado automático y planes de recursos

- Reglas de escalado automático: puede aumentar o disminuir los nodos de tarea en función de las cargas de clúster en tiempo real. El escalado automático se activará cuando cambie el volumen de datos, pero puede haber algún retraso.
- Planes de recursos: si el volumen de datos cambia periódicamente, puede crear planes de recursos para cambiar el tamaño del clúster antes de que cambie el volumen de datos, evitando así un retraso en el aumento o la disminución de recursos.

Tanto las reglas de escalado automático como los planes de recursos pueden desencadenar escalado automático. Puede configurar ambos o configurar uno de ellos. La configuración de planes de recursos y reglas de escalado automático mejora la escalabilidad del nodo del clúster para hacer frente a picos de volumen de datos ocasionalmente inesperados.

En algunos escenarios de servicio, los recursos deben reasignarse o la lógica de servicio debe modificarse después de escalar o reducir el clúster. Si se escala o reduce manualmente un clúster, puede iniciar sesión en los nodos del clúster para reasignar recursos o modificar la lógica del servicio. Si utiliza escalado automático, MRS le permite personalizar secuencias de comandos de automatización para la reasignación de recursos y la modificación de la lógica de servicio. Los scripts de automatización pueden ejecutarse antes y después del escalado automático y adaptarse automáticamente a los cambios de carga de servicio, todo lo cual elimina las operaciones manuales. Además, los scripts de automatización se pueden personalizar completamente y ejecutar en varios momentos, lo que puede satisfacer sus requisitos personalizados y mejorar la flexibilidad de escalado automático.

Beneficios para el cliente

El escalado automático MRS proporciona los siguientes beneficios:

- Reducción de costes

  Las empresas no analizan los datos todo el tiempo, sino que realizan un análisis de datos por lotes en un período de tiempo específico, por ejemplo, a las 03:00 a.m. El análisis por lotes puede tardar sólo dos horas.

  La función de escalado automático permite a las empresas agregar nodos para el análisis por lotes y libera automáticamente los nodos después de completar el análisis, minimizando los costos.

- Cumplir los requisitos de consultas instantáneas

  Las empresas suelen encontrarse con tareas de análisis instantáneo, por ejemplo, informes de datos para respaldar la toma de decisiones empresariales. Como resultado, el consumo de recursos aumenta bruscamente en un corto período de tiempo. Con la función de escalado automático, se pueden agregar nodos de cómputo para el análisis emergente de big data, evitando una avería del servicio debido a recursos de cómputo insuficientes. De esta manera, no es necesario comprar recursos adicionales. Después de que termine la emergencia, MRS libera automáticamente los nodos.

- Centrarse en el negocio principal

  Es difícil para los desarrolladores determinar el consumo de recursos en la plataforma de desarrollo secundario de big data debido a las complejas condiciones de análisis de consultas (como la clasificación global, el filtrado y la fusión) y la complejidad de los datos, por ejemplo, la incertidumbre de los datos incrementales. Como resultado, estimar el volumen de cálculo es difícil. La función de escalado automático de MRS permite a los desarrolladores centrarse en el desarrollo de servicios sin necesidad de estimación de recursos.

7.9.4 Creación de nodos de tarea

Introducción a las funciones

Los nodos de tarea solo se pueden crear y usar para computación. No almacenan datos persistentes y son la base para implementar el escalado automático.

Beneficios para el cliente

Cuando MRS se utiliza solo como un recurso informático, los nodos de tarea se pueden utilizar para reducir costos y facilitar el escalado de nodos de clúster, cumpliendo de manera flexible los requisitos de los usuarios para aumentar o disminuir las capacidades informáticas de clúster.

Escenarios de aplicación

Cuando el cambio de volumen de datos es pequeño en un clúster pero las capacidades de procesamiento de servicios del clúster necesitan mejorarse notablemente y temporalmente, agregue nodos de Task para abordar las siguientes situaciones:

- El número de servicios temporales aumenta, por ejemplo, el procesamiento de informes al final del año.
- Las tareas a largo plazo deben completarse en poco tiempo, por ejemplo, algunas tareas de análisis urgentes.

7.9.5 Escalamiento de las especificaciones del nodo Master

MRS proporciona a Manager para gestionar clústeres y servicios en los clústeres, como NameNodes de HDFS, ResourceManagers de Yarn y los servicios de gestión de Manager de MRS, que se despliega en el nodo Master de los clústeres.

Con la implementación de nuevos servicios, una escala de clúster aumenta continuamente, y los nodos Master soportan más y más cargas. Los usuarios empresariales se enfrentan al problema de que las cargas de CPU son demasiado altas y el uso de memoria excede el umbral. Generalmente, en un clúster de big data local, necesita migrar datos y comprar hardware con configuraciones avanzadas para ampliar las especificaciones del nodo Master. MRS aprovecha las ventajas de los servicios en la nube para que pueda ampliar las especificaciones del nodo Master con un solo clic. Durante la ampliación, el modo HA activo/en espera de los nodos Master asegura que los servicios existentes no se interrumpan.

Para obtener detalles sobre cómo ampliar las especificaciones del nodo maestro, consulte **Escalamiento de especificaciones de nodo Master**.

7.9.6 Aislamiento de un host

Al detectar que un host es anormal o defectuoso y no puede proporcionar servicios o afecta al rendimiento del clúster, puede excluir el host de los nodos disponibles en el clúster temporalmente para que el cliente pueda tener acceso a otros nodos disponibles. En los escenarios en los que se van a instalar parches en un clúster, también puede excluir un nodo especificado de la instalación de parches. Solo se pueden aislar nodos que no sean de gestión.

Después de aislar un host, todas las instancias de rol del host se detendrán y no podrá iniciar, detener o configurar el host y todas las instancias del host. Además, después de aislar un host, no se pueden recopilar ni mostrar estadísticas sobre el estado de supervisión y los datos métricos del hardware y las instancias en el host.

7.9.7 Gestión de etiquetas

Las etiquetas son identificadores de clúster. Agregar etiquetas a los clústeres puede ayudarlo a identificar y gestionar los recursos del clúster. Al asociarse con Tag Management Service (TMS), MRS permite a los usuarios con una gran cantidad de recursos en la nube etiquetar recursos en la nube, buscar rápidamente recursos en la nube con el mismo atributo de etiqueta y realizar operaciones de gestión unificadas como revisión, modificación y eliminación, facilitando la gestión unificada de clústeres de big data y otros recursos en la nube.

Puede agregar un máximo de 10 etiquetas a un clúster al crear el clúster o agregarlas en la página de detalles del clúster creado.

7.10 Clúster O&M

Gestión de alarma

MRS puede monitorear clústeres de big data en tiempo real e identificar el estado del sistema basado en alarmas y eventos. Además, MRS le permite personalizar los umbrales de monitoreo y alarma para centrarse en el estado de salud de cada métrica. Cuando los datos de monitorización alcanzan el umbral de alarma, el sistema activa una alarma.

MRS también puede interconectarse con el sistema de servicio de mensajes del servicio de notificación de mensajes simples (SMN) de Huawei Cloud para enviar información de alarma a los usuarios por mensaje SMS o correo electrónico. Para obtener más información, consulte **Notificación de mensaje**.

Gestión de parches

MRS admite operaciones de parches de clúster y lanzará parches para componentes de big data de código abierto de manera oportuna. En la página de gestión de clústeres de MRS, puede ver la información de la versión de parches relacionada con los clústeres en ejecución, incluida la descripción detallada de los problemas y los impactos resueltos. Puede determinar si desea instalar un parche en función del estado de ejecución del servicio. La instalación de parches con un solo clic no implica ninguna intervención manual y no causará interrupción del servicio a través de la instalación continua, lo que garantiza la disponibilidad a largo plazo de los clústeres.

MRS puede mostrar el proceso detallado de instalación de parches. La gestión de parches también admite la desinstalación y la reversión de parches.

NOTA

MRS 3.x o posterior no admite la gestión de parches en la consola de gestión.

Soporte de O&M

Los recursos de clúster proporcionados por MRS pertenecen a los usuarios. Generalmente, cuando se requiere el soporte del personal de O&M para la solución de problemas de un clúster, el personal de O&M no puede acceder directamente al clúster. Para ofrecer un mejor servicio a los clientes, MRS proporciona los dos métodos siguientes para mejorar la eficiencia de la comunicación durante la localización de fallas:

- Uso compartido de registros: puede iniciar el uso compartido de registros en la consola de gestión de MRS para compartir un ámbito de registro especificado con el personal de

O&M, de modo que el personal de O&M pueda localizar fallas sin tener acceso al clúster.

- Autorización O&M: Si se produce un problema al utilizar un clúster MRS, puede iniciar la autorización O&M en la consola de gestión de MRS. El personal de O&M puede ayudarlo a localizar rápidamente el problema, y puede revocar la autorización en cualquier momento.

Comprobación de estado

MRS proporciona una inspección automática de los entornos de funcionamiento del sistema para que pueda comprobar y auditar el estado de funcionamiento del sistema con un solo clic, lo que garantiza un funcionamiento correcto del sistema y reduce los costos de operación y mantenimiento del sistema. Después de ver los resultados de la inspección, puede exportar informes para archivarlos y analizar fallas.

7.11 Notificación de mensaje

Introducción a las funciones

Las siguientes operaciones se realizan a menudo durante la ejecución de un clúster de big data:

- Los clústeres de big data a menudo cambian, por ejemplo, escalamiento horizontal y vertical del clúster.
- Cuando un volumen de datos de servicio cambia bruscamente, se activa el escalado automático.
- Una vez que se detienen los servicios relacionados, es necesario detener un clúster de big data.

Para notificarle inmediatamente de las operaciones exitosas, la indisponibilidad del clúster y los fallos de los nodos, MRS utiliza la notificación de mensaje simple (SMN) para enviarle notificaciones a través de SMS y correos electrónicos, lo que facilita el mantenimiento.

Beneficios para el cliente

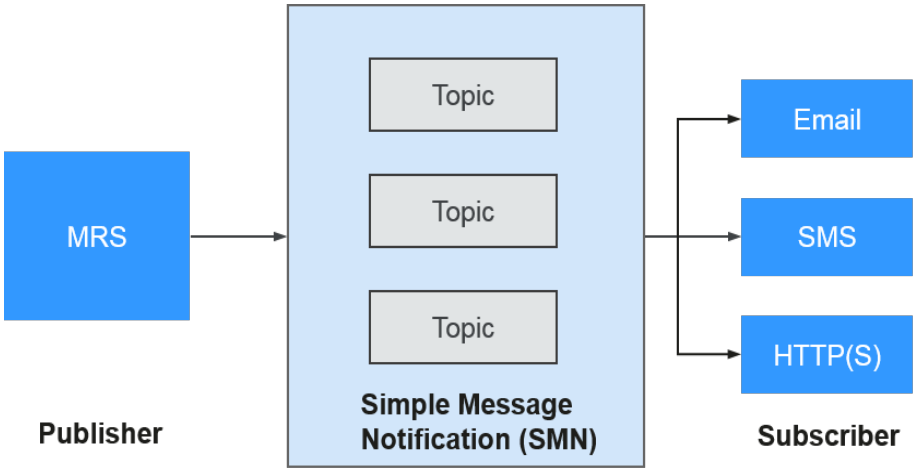
Después de configurar SMN, puede recibir el estado del clúster MRS, actualizaciones y alarmas de componentes a través de SMS o correos electrónicos en tiempo real. MRS envía monitoreo en tiempo real y notificación de alarma para ayudarle a realizar operaciones de operación fácilmente y a implementar eficientemente servicios de big data.

Descripción de la función

MRS utiliza SMN para proporcionar una suscripción y notificación de mensaje de uno a varios a través de una variedad de protocolos.

Puede crear un tema y configurar políticas de tema para controlar los permisos de editor y suscriptor en el tema. MRS envía mensajes de clúster al tema para el que tiene permiso para publicar mensajes. Entonces, todos los suscriptores que se suscriben al tema pueden recibir actualizaciones del clúster y alarmas de componentes a través de SMS y correos electrónicos.

Figura 7-1 Proceso de implementación



8 Seguridad

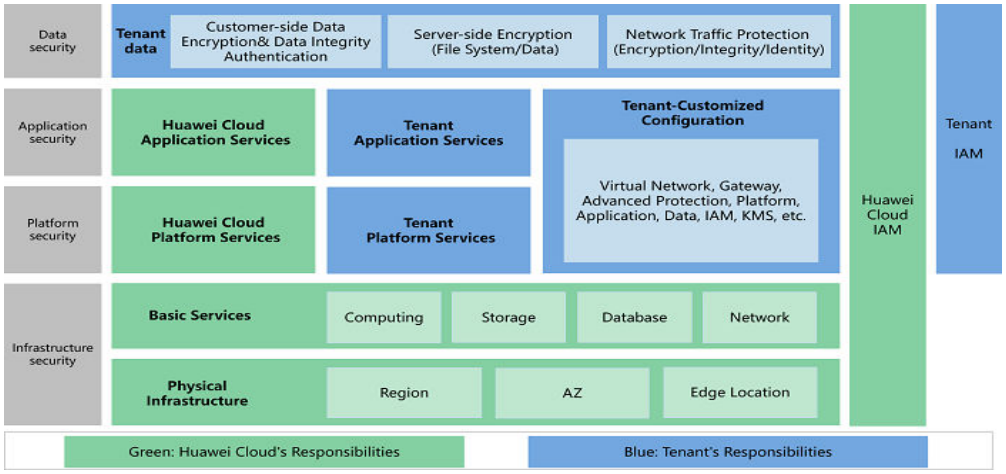
8.1 Responsabilidades compartidas

Huawei Cloud garantiza que su compromiso con la ciberseguridad nunca se verá compensado por la consideración de intereses comerciales. Para hacer frente a los desafíos emergentes de seguridad en la nube y a las amenazas y ataques generalizados de seguridad en la nube, Huawei Cloud crea un sistema integral de garantía de seguridad de servicios en la nube para diferentes regiones e industrias basado en las ventajas únicas de software y hardware, las leyes, las regulaciones, los estándares de la industria y el ecosistema de seguridad de Huawei.

Figura 8-1 ilustra las responsabilidades compartidas por Huawei Cloud y los usuarios.

- Huawei Cloud: garantiza la seguridad de los servicios en la nube y proporciona nubes seguras. Las responsabilidades de seguridad de Huawei Cloud incluyen garantizar la seguridad de nuestros servicios IaaS, PaaS y SaaS, así como los entornos físicos de los centros de datos de Huawei Cloud donde nuestros IaaS, PaaS, y los servicios SaaS operan. Huawei Cloud es responsable no solo de las funciones de seguridad y el rendimiento de nuestra infraestructura, servicios en la nube y tecnologías, sino también de la seguridad general de la nube y, en el sentido más amplio, del cumplimiento de seguridad de nuestra infraestructura y servicios.
- Inquilino: utiliza la nube de forma segura. Los inquilinos de Huawei Cloud son responsables de la gestión segura y efectiva de la seguridad interna, así como de las configuraciones personalizadas del inquilino de los servicios en la nube, incluidos IaaS, PaaS y SaaS. Esto incluye, entre otros, sistemas operativos como redes virtuales, máquinas virtuales huésped y host de máquinas virtuales, firewall virtual, API Gateway y servicios de seguridad avanzados, todo tipo de servicios en la nube, datos de inquilinos, cuentas de identidad y gestión de claves.

Figura 8-1 Modelo de responsabilidad de seguridad compartida de Huawei Cloud



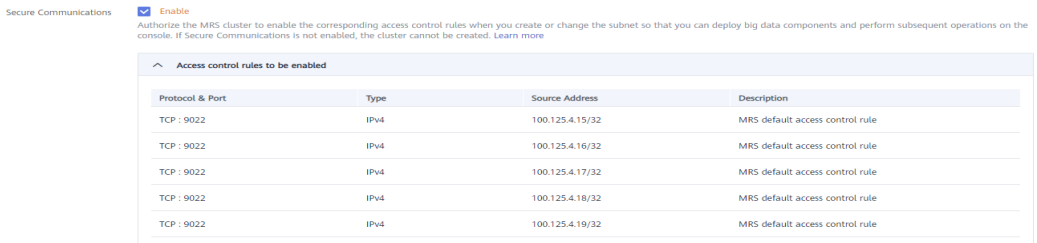
8.2 Identificación y gestión de activos

Comunicaciones seguras

En un clúster MRS, puede aprovisionar, gestionar y usar componentes de big data a través de la consola de gestión. Los componentes de Big Data se implementan en las VPC de los usuarios. Para permitir que la consola de MRS acceda directamente a los componentes de big data, debe habilitar las reglas de grupo de seguridad correspondientes después de conceder la autorización. Este proceso de autorización se llama comunicaciones seguras.

Es necesario habilitar las comunicaciones seguras al crear un clúster, como se muestra en **Figura 8-2**.

Figura 8-2 Comunicaciones seguras

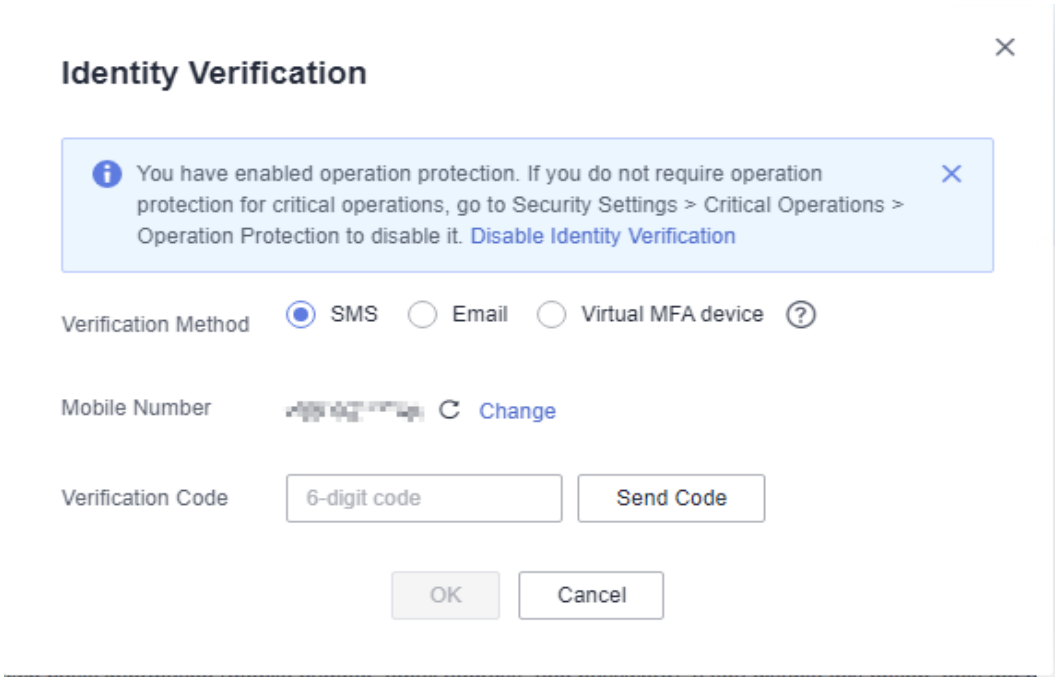


Se recomienda habilitar las reglas de grupo de seguridad solo para las direcciones IP de confianza. Tenga cuidado al usar 0.0.0.0/0 como la dirección de origen del grupo de seguridad.

Protección para operaciones críticas

MRS proporciona protección para operaciones críticas. Si ha habilitado la protección de operación (para obtener más información, consulte **Protección de operación crítica** de IAM), introduzca el código de verificación obtenido mediante el método de verificación que ha seleccionado (como se muestra en **Figura 8-3**) para evitar riesgos y pérdidas causadas por operaciones incorrectas.

Figura 8-3 Verificación de identidad



8.3 Autenticación de identidad y control de acceso

Autenticación de identidad

MRS admite el protocolo de seguridad Kerberos. FusionInsight MRS utiliza LDAP para el sistema de gestión de cuentas y realiza autenticación de seguridad en la información de la cuenta a través de Kerberos.

Para obtener más información sobre el mecanismo de autenticación de seguridad de Kerberos, consulte **Principios y mecanismos de autenticación de seguridad**.

Control de acceso

MRS proporciona dos modelos de control de acceso: control de acceso basado en roles y control de acceso basado en políticas. Para obtener más información, consulte **Modelo de permiso**.

- **Control de acceso basado en rol**

  Mediante el empleo de un sistema de autenticación unificado de usuario y basado en roles y cumpliendo con el modelo de control de acceso basado en cuentas/funciones (RBAC), MRS implementa la gestión de permisos basada en roles y la gestión de autorización de usuarios por lotes. También proporciona la capacidad de inicio de sesión único (SSO) para ofrecer una gestión y autenticación unificadas para los usuarios del sistema MRS de FusionInsight y los usuarios de componentes. Para obtener más información sobre el mecanismo, consulte **Mecanismo de permiso**.

- **Control de acceso basado en política**
  - Autenticación de Ranger

    MRS admite la autenticación de Ranger. Para un clúster MRS en modo de seguridad, la autenticación de Ranger está habilitada de forma predeterminada. Para

un clúster normal con el servicio Ranger instalado, Ranger admite el control de permisos en recursos de componentes basados en usuarios de sistema operativo.

Para obtener más información sobre las políticas de autenticación de Ranger, consulte **Políticas de verificación de permisos**.

- Autenticación de grano fino para clústeres desacoplados de cálculo de almacenamiento de OBS

  Si desea realizar un control de permisos detallado en recursos de OBS en clústeres desacoplados de cálculo de almacenamiento de OBS, MRS le proporciona una solución de control de permisos detallado basada en la agencia IAM.

  Para obtener más información, consulte **Configuración de permisos de grano fino para el acceso de varios usuarios de MRS a OBS**.

8.4 Tecnologías de protección de datos

Integridad de los datos

Los datos se verifican para garantizar su integridad durante el almacenamiento y la transmisión.

Los datos de usuario de MRS se almacenan en HDFS, que utiliza CRC32C para verificar los datos. HDFS también admite la verificación CRC32, que es mucho más rápida que CRC32C. Los DataNodes HDFS almacenan los datos verificados. Si detectan que los datos transmitidos desde el cliente están incompletos, notifican la excepción al cliente y notifican al cliente la retransmisión de datos. El cliente comprueba la integridad de los datos cuando lee datos de un DataNode. Si los datos están incompletos, el cliente leerá los datos de otro DataNode.

Confidencialidad de los datos

Basado en Apache Hadoop, el sistema de archivos distribuido de FusionInsight MRS proporciona almacenamiento cifrado de archivos para evitar que los datos confidenciales se almacenen en texto plano, mejorando la seguridad de los datos.

Las aplicaciones necesitan cifrar solo los datos confidenciales especificados. Los servicios no se ven afectados durante el cifrado y descifrado. Además del cifrado de datos del sistema de archivos, Hive proporciona cifrado de columna (consulte **Uso de la función de cifrado de columna Hive**). Los datos confidenciales se pueden cifrar y almacenar después de especificar un algoritmo de encriptación durante la creación de la tabla. HBase admite el encriptación de HFiles y WALs (consulte **Cifrado de HFile y WAL**). Puede configurar los algoritmos AES y SMS4 para cifrarlos.

Seguridad de transmisión de datos

En un clúster MRS, se admite el cifrado de HTTPS para el acceso a través de canales de web. La comunicación RPC admite la autenticación SASL y el cifrado de datos mediante claves simétricas. La configuración de transmisión cifrada de cada componente es la siguiente:

- Configuración de transmisión encriptada HDFS: consulte **Configuración del cifrado de datos HDFS durante la transmisión**.
- Configuración de transmisión encriptada de Kafka: Consulte **Configuración del cifrado de datos de Kafka durante la transmisión**.
- Configuración de transmisión cifrada de Flume: consulte **Configuración de transmisión cifrada**.

- Configuración de transmisión cifrada de Flink: Consulte **Transmisión cifrada** en **Autenticación y Cifrado**.

Copia de respaldo de datos y recuperación ante desastres

- Recuperación ante desastres (DR): MRS admite copias de respaldo de datos en Huawei Cloud OBS y ofrece una alta confiabilidad entre regiones.
- Copia de respaldo: FusionInsight MRS puede hacer copias de respaldo de los metadatos de OMS, Kafka, DBService y NameNodes así como de los datos de servicio de HDFS, HBase y Hive.

Para obtener más información, consulte **Introducción al gestión de copia de respaldo y recuperación**.

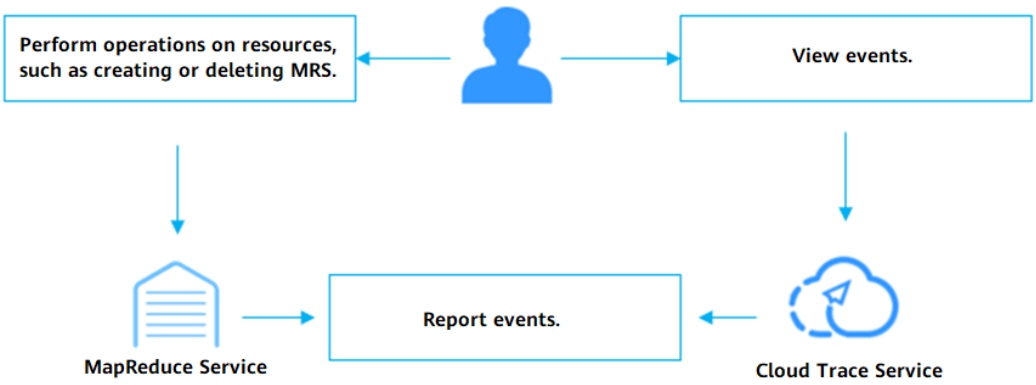
8.5 Auditoría y registro

Auditoría

Cloud Trace Service (CTS) registra los registros de operaciones de MRS en la consola de gestión, como la creación o eliminación de clústeres de MRS. CTS es un servicio de auditoría de registros destinado a la seguridad en la nube. Registra las operaciones en los recursos de la nube en su cuenta. Puede utilizar los registros generados por CTS para realizar análisis de seguridad, realizar un seguimiento de los cambios de recursos, auditar el cumplimiento y localizar fallas.

Después de habilitar CTS y configurar un rastreador, CTS puede registrar la gestión y los rastros de datos de MRS para su auditoría.

Figura 8-4 Registro de eventos de MRS en CTS



FusionInsight Manager proporciona la función de auditoría para registrar las operaciones del usuario en el Administrador de clústeres. En la página **Audit**, los administradores pueden ver los registros de operaciones históricos de los usuarios en Manager. Los registros se pueden usar para localizar fallas y determinar responsabilidades en eventos de seguridad. Para obtener más información sobre la página **Audit**, consulte **Descripción de auditoría**. Los registros de auditoría del FusionInsight Manager se almacenan en la base de datos de forma predeterminada. Si los registros de auditoría se conservan durante mucho tiempo, el espacio en disco del directorio de datos puede resultar insuficiente. Para almacenar registros de

auditoría en otro servidor de archivado, los administradores pueden establecer los parámetros de volcado necesarios para volcar automáticamente estos registros. Esto facilita la gestión de los registros de auditoría. Para obtener más información acerca de cómo volcar registros de auditoría, consulte **Configuración del volcado del registro de auditoría**.

Registro

Los registros de todos los componentes de un clúster de MRS (por ejemplo, todos los registros de HDFS) se pueden recopilar conectando hosts con Log Tank Service (LTS). LTS recopila datos de registro de hosts y servicios en la nube. Al analizar y procesar cantidades masivas de registros de manera eficiente, segura y en tiempo real, LTS proporciona información útil para optimizar la disponibilidad y el rendimiento de los servicios y aplicaciones en la nube. También le ayuda a realizar eficientemente la toma de decisiones en tiempo real, la gestión de O&M de dispositivos y el análisis de tendencias de servicio. Para obtener más información sobre la interconexión, consulte **Cómo interconecto MRS con LTS**.

FusionInsight Manager también admite la búsqueda en línea de registros de componentes para la localización de fallas y otros escenarios. Para obtener más información, consulte **Registro de búsqueda en línea**. Además, FusionInsight Manager le permite exportar los registros generados por todas las instancias de cada rol de servicio en lotes, sin necesidad de iniciar sesión manualmente en cada nodo. Para obtener más información, consulte **Descarga de registro**.

8.6 Resiliencia del servicio

Despliegue de DR entre AZ

El plano de gestión de MRS proporciona la capacidad de DR cruzada AZ de doble clúster. Puede desplegar un clúster DR de MRS homogéneo en otro zona de disponibilidad. Si el clúster de producción no proporciona servicios de lectura y escritura debido a un desastre natural o a fallas internas del clúster, el clúster de recuperación ante desastres se convertirá en el clúster de producción para garantizar la continuidad del servicio.

8.7 Monitoreo de riesgos de seguridad

Manager del servicio MRS proporciona capacidades de monitoreo a nivel de clúster para ayudarlo a monitorear el estado de salud de los componentes y nodos de big data en los clústeres. También proporciona la capacidad de notificación de alarma para que pueda obtener información sobre las métricas y el estado de salud de los clústeres de MRS en tiempo real.

8.8 Gestión de actualizaciones

Actualización de contraseña

MRS admite la actualización de la contraseña. Se recomienda cambiar la contraseña periódicamente de acuerdo con las siguientes directrices para mejorar la seguridad del sistema:

- Cambiar la contraseña de un usuario del sistema: consulte **Cambio de la contraseña del usuario admin** y **Cambio de la contraseña de usuario del sistema operativo**.

- **Cambio de la contraseña de un usuario interno del sistema**
- **Cambio de la contraseña de un usuario de base de datos**

Actualización de certificado

Tanto el certificado de CA como el certificado HA de MRS pueden reemplazarse. Puede reemplazar los certificados predeterminados de un clúster de acuerdo con las siguientes instrucciones:

- El certificado de MRS CA se utiliza para cifrar datos durante la comunicación entre el cliente y el servidor de un componente para garantizar la seguridad de la comunicación. Para obtener más información acerca de cómo reemplazar un certificado de CA, consulte **Reemplazar el certificado de CA**.
- El certificado de HA se utiliza para cifrar datos durante la comunicación entre los procesos activos/en espera y los procesos HA para garantizar la seguridad de la comunicación. Para obtener más información acerca de cómo reemplazar un certificado de HA, consulte **Reemplazar certificados de HA**.

8.9 Mejoras de seguridad

Mejoras de Tomcat

Durante la instalación y el uso del Administrador de FusionInsight, las siguientes funciones de Tomcat se mejoran sobre la base de la versión de código abierto:

- Tomcat se ha actualizado a una versión oficial estable.
- Los permisos en los directorios bajo aplicaciones se establecen en **500** y se admite el permiso de escritura en algunos directorios.
- El paquete de instalación de Tomcat se elimina automáticamente después de instalar el software del sistema.
- La función de despliegue automático está deshabilitada para proyectos en directorios de aplicaciones. Solo se implementan los proyectos **web**, **cas** y **client**.
- Algunos métodos **http** no utilizados están deshabilitados, evitando ataques que se pueden lanzar mediante el uso de los métodos **http**.
- El puerto de apagado predeterminado y el comando del servidor de Tomcat se cambian para evitar que los piratas informáticos cierren el servidor y ataquen el servidor y las aplicaciones.
- Para garantizar la seguridad, se cambia el valor de **maxHttpHeaderSize**, lo que permite a los administradores del servidor controlar las solicitudes anormales de los clientes.
- El archivo de descripción de la versión de Tomcat se modifica después de instalar Tomcat.
- Para evitar la divulgación de información de Tomcat, los atributos del servidor de Connector se modifican para que los atacantes no puedan obtener información sobre el servidor.
- Los permisos en los archivos y directorios de Tomcat, como los archivos de configuración, archivos ejecutables, directorios de registro y carpetas temporales, están bajo control.
- El reciclaje de fachadas de sesión está desactivado para evitar fugas de solicitudes.

- LegacyCookieProcessor se utiliza como CookieProcessor para evitar la filtración de datos sensibles en las cookies.

Mejora de LDAP

LDAP se endurece de la siguiente manera después de instalar un clúster:

- En el archivo de configuración LDAP, la contraseña de la cuenta de administrador se cifra mediante SHA. Después de actualizar el OpenLDAP a 2.4.39 o posterior, los datos se sincronizan automáticamente entre los nodos LDAP activo y en espera mediante el mecanismo externo SASL, que impide la divulgación de la contraseña.
- El servicio LDAP del clúster admite el protocolo SSLv3 de forma predeterminada, que se puede utilizar de forma segura. Cuando OpenLDAP se actualiza a 2.4.39 o posterior, LDAP utiliza automáticamente TLS1.0 o posterior para evitar riesgos de seguridad desconocidos.

Otra mejora de seguridad

Para obtener más información sobre otras guías de refuerzo de seguridad, consulte **Mejora de seguridad**.

9 Restricciones

Antes de usar MRS, asegúrese de haber leído y comprendido las siguientes restricciones.

- Los clústeres de MRS deben crearse en subredes de VPC.
- Se aconseja utilizar cualquiera de los siguientes navegadores para acceder a MRS:
  - Google Chrome: 36.0 o posterior
  - Internet Explorer: 9.0 o posterior

    Si utiliza Internet Explorer 9.0, es posible que no pueda iniciar sesión en la consola de gestión de MRS porque el **Administrator** de usuario está deshabilitado de forma predeterminada en algunos sistemas Windows, como Windows 7 Ultimate. Internet Explorer selecciona automáticamente un usuario del sistema para su instalación. Como resultado, Internet Explorer no puede acceder a la consola de gestión. Vuelva a instalar Internet Explorer 9.0 o posterior (recomendado) o ejecute Internet Explorer 9.0 como **Administrator** de usuario.
- Cuando crea un clúster MRS, puede seleccionar **Auto create** en la lista desplegable de **Security Group** para crear un grupo de seguridad o seleccionar un grupo de seguridad existente. Después de crear el clúster MRS, no elimine ni modifique el grupo de seguridad usado. De lo contrario, puede producirse una excepción de clúster.
- Para evitar el acceso ilegal, solo asigne permiso de acceso a los grupos de seguridad utilizados por MRS cuando sea necesario.
- No realice las siguientes operaciones porque causarán excepciones de clúster:
  - Apagar, reiniciar o eliminar los nodos del clúster de MRS mostrados en ECS, cambiar o reinstalar su sistema operativo o modificar sus especificaciones.
  - Eliminar los procesos, aplicaciones o archivos existentes en los nodos del clúster.
  - Eliminación de nodos de clúster de MRS. Los nodos eliminados se seguirán cobrando.
- Si se produce una excepción de clúster cuando no se han realizado operaciones incorrectas, póngase en contacto con los ingenieros de soporte técnico. Ellos le pedirán su contraseña y luego realizar la solución de problemas.
- Mantenga la contraseña inicial para iniciar sesión en el nodo maestro correctamente porque MRS no lo guardará. Utilice una contraseña compleja para evitar ataques maliciosos.
- Los clústeres de MRS todavía se cobran durante las excepciones. Póngase en contacto con los ingenieros de soporte técnico para gestionar las excepciones del clúster.

- Planifique discos de nodos de clúster según los requisitos de servicio. Si desea almacenar un gran volumen de datos de servicio, agregue discos EVS o espacio de almacenamiento para evitar que el espacio de almacenamiento insuficiente afecte a la ejecución del nodo.
- Los nodos del clúster almacenan únicamente los datos de servicio de los usuarios. Los datos no de servicio se pueden almacenar en el OBS u otros nodos de ECS.
- Los nodos del clúster solo ejecutan programas de clúster de MRS. Otras aplicaciones de cliente o programas de servicio de usuario se despliega en nodos de ECS separados.
- Para ampliar la capacidad de almacenamiento de los nodos (incluidos master, core, y task) en un clúster de MRS, compre discos nuevos y conéctelos a los nodos. Para obtener más información, consulte **Ampliación de la capacidad para un disco de EVS en uso**.
- La capacidad (incluidas las capacidades de almacenamiento y computación) de un clúster de MRS se puede ampliar agregando nodos de núcleo o de tarea.
- Si el clúster se sigue utilizando para ejecutar tareas o modificar configuraciones después de que se detenga un nodo de master en el clúster y otros nodos de master en el clúster se detienen antes de que se inicie el nodo de master detenido después de la ejecución de la tarea o la modificación de la configuración, los datos pueden perderse debido a una conmutación activa/en espera. En este escenario, una vez ejecutada la tarea o modificada la configuración, inicie el nodo de master que se ha detenido y, a continuación, detenga todos los nodos. Si se han detenido todos los nodos del clúster, inícielos en el orden inverso del cierre del nodo.
- La conmutación del programador de Capacity y Superior se completa cuando se utiliza el clúster de MRS, mientras que la sincronización de configuración no se completa. Configure la sincronización de nuevo en función del nuevo programador si es necesario.

10 Facturación

Los precios de MapReduce Service (MRS) son simples y predecibles. Puede seleccionar un modo de facturación de pago por uso o una suscripción anual/mensual dependiendo de lo que sea más económico para usted. El precio total de un clúster MRS se calculará automáticamente para que pueda comprar un clúster con un solo clic.

Conceptos de facturación:

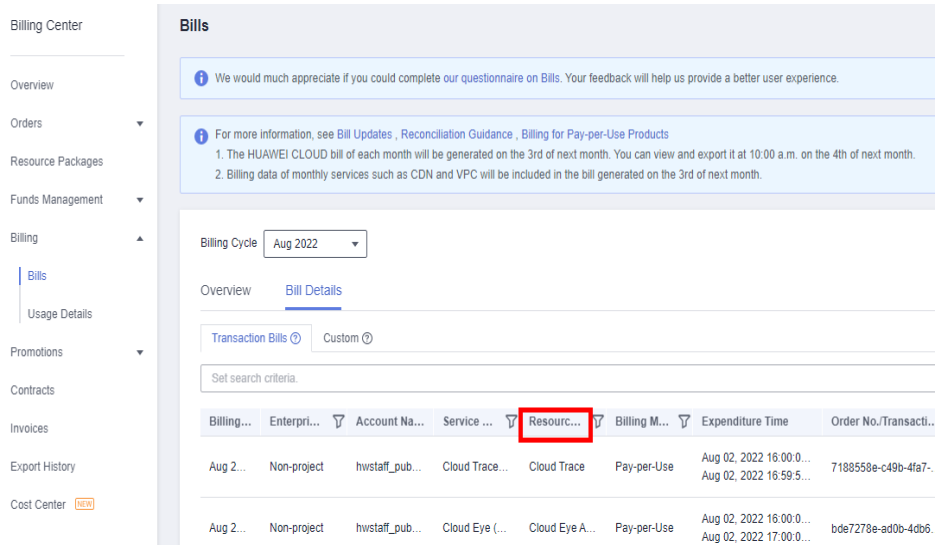
El precio de un clúster de MRS consta de dos partes:

- Tarifa de gestión del MRS

NOTA

Puede consultar la tarifa de gestión de MRS detallada iniciando sesión en el Centro de facturación, seleccionando **Billing** > **Bills** y filtrando las tarifas de gestión.

Figura 10-1 Consulta de la tarifa de gestión de MRS



- Si **Cluster Type** tiene un valor **LTS**, filtre las tarifas de gestión por **MRS-LTS Service Fee**.
- Si **Cluster Type** tiene un valor **Normal**, filtre las tarifas de gestión de la siguiente manera:
  - **MapReduce Service VM** para clústeres comprados en junio de 2022 o antes
  - **MRS-BASIC Service Fee** para clústeres comprados después de junio de 2022

- Tarifas de recursos de infraestructura IaaS, incluidos Elastic Cloud Server (ECS), Elastic Volume Service (EVS), elastic IP (EIP) y ancho de banda

Para obtener más información sobre la tarifa de gestión de MRS, consulte Detalles de precio de producto. Puede utilizar la calculador de precio de MRS para obtener rápidamente un precio estimado de un grupo con las especificaciones que seleccione.

El clúster de MRS terminado o cancelado ya no se factura.

Modos de facturación

Antes de usar MRS, debe comprar un clúster de MRS. Actualmente, MRS ofrece dos modos de facturación:

- Anual/Mensual: puede pagar por clústeres por año o mes. La duración mínima es de 1 mes y la máxima es de 1 año.
- Pago por uso: los nodos se facturan por la duración real del uso, con un ciclo de facturación de una hora.

Cambio del modo de facturación

Antes de suscribirse a MRS, elija las instancias de nodo principal y principal que mejor se adapten a sus necesidades. MRS proporciona los siguientes métodos para cambiar la configuración del clúster después de iniciar un clúster.

- **Configurar nodo de tarea**: Agregar nodos de tarea. Para obtener más información, consulte **Operaciones realcionadas** en **Escalado horizontal manual de un clúster**.
- **Escalar**: Agregar manualmente los nodos de Core o de Task. Para obtener más información, consulte **Escalado horizontal manual de un clúster**.
- **Auto Scaling**: el número de nodos de un clúster se puede ajustar automáticamente en función del volumen de datos de servicio para aumentar o disminuir los recursos. Para obtener más información, consulte **Configuración de reglas de escalado automático**.

Si los métodos de cambio de configuración proporcionados por MRS no cumplen con sus requisitos, puede crear un clúster de nuevo y migrar datos al clúster para realizar el cambio de configuración del clúster.

Renovación

Para renovar la suscripción, vaya a la página **Renovaciones**.

Pago atrasado

El pago atrasado no se aplica a los clústeres suscritos anualmente o mensuales.

En el modo de pago por uso, las tarifas de clúster se deducen cada hora. Si el saldo de su cuenta es insuficiente para pagar los gastos ocurridos en la última hora, su cuenta estará en mora, y los clústeres MRS tienen un **período de retención**. Si los clústeres se renuevan dentro del período de retención, estarán disponibles y se cobrarán a partir de la fecha de vencimiento original.

Se recomienda renovar el clúster lo antes posible si el clúster está en mora. De lo contrario, se restringen las siguientes operaciones:

- Creación de clústeres
- Escalamiento horizontal de un clúster
- Escalamiento en un clúster
- Adición de un nodo Task
- Escalamiento de las especificaciones del nodo Master

Caducidad

- El vencimiento no se aplica a los clústeres de pago por uso.
- Si su suscripción anual o mensual caduca, el clúster entrará en un **período de retención**. Durante el período de gracia y el período de retención, no puede realizar operaciones en el clúster en la consola de gestión de MRS, no se puede invocar a las API relacionadas y se detendrán las operaciones de O&M, como la supervisión automática y los informes de alarma. Si su suscripción no se renueva al final del período de retención, los servicios del clúster se cancelarán y los datos del sistema se eliminarán de forma permanente.

11 Gestión de permisos

Si necesita asignar diferentes permisos a los empleados de su empresa para acceder a sus recursos de MRS en Huawei Cloud, IAM es una buena opción para la gestión de permisos detallada. IAM proporciona autenticación de identidad, gestión de permisos y control de acceso, lo que le ayuda a proteger el acceso a sus recursos de Huawei Cloud.

Con IAM, puede crear usuarios de IAM bajo su cuenta de Huawei Cloud y asignar permisos a estos usuarios para controlar su acceso a recursos específicos. Por ejemplo, algunos desarrolladores de software de su empresa necesitan usar recursos de MRS, pero no deben eliminar los clústeres de MRS ni realizar operaciones de alto riesgo. Para lograr este objetivo, puede crear usuarios de IAM para los desarrolladores de software y concederles solo los permisos necesarios para usar los recursos del clúster de MRS.

Si su cuenta de Huawei Cloud no requiere usuarios individuales de IAM para la gestión de permisos, omita esta sección.

IAM es gratuito. Usted paga solo por los recursos que utiliza. Para obtener más información acerca de IAM, consulte **Descripción de servicio IAM**.

Descripción de permiso de MRS

De forma predeterminada, los nuevos usuarios de IAM no tienen permisos. Para asignar permisos a un usuario, agregue el usuario a uno o más grupos y asigne políticas o roles de permisos a estos grupos. A continuación, el usuario hereda los permisos de los grupos de los que es miembro y puede realizar operaciones específicas en servicios en la nube basadas en los permisos.

MRS es un servicio a nivel de proyecto implementado y accedido en regiones físicas específicas. Para asignar permisos a un grupo de usuarios, especifique **Scope** como **Region-specific projects** y seleccione proyectos en la región correspondiente para que los permisos surtan efecto. Si se selecciona **All projects**, los permisos surtirán efecto para el grupo de usuarios en todos los proyectos específicos de la región. Al acceder a MRS, los usuarios necesitan cambiar a una región en la que han sido autorizados para usar el servicio MRS.

Puede conceder permisos a los usuarios mediante roles y políticas.

- Roles: Un tipo de mecanismo de autorización de grano grueso que define permisos relacionados con las responsabilidades del usuario. Este mecanismo proporciona solo un número limitado de roles de nivel de servicio para la autorización. Al usar roles para conceder permisos, también debe asignar otros roles de los que dependen los permisos

para que surtan efecto. Sin embargo, los roles no son una opción ideal para la autorización detallada y el control de acceso seguro.

- Políticas: Un tipo de mecanismo de autorización detallado que define los permisos necesarios para realizar operaciones en recursos de nube específicos bajo ciertas condiciones. Este mecanismo permite una autorización más flexible basada en políticas, cumpliendo los requisitos para un control de acceso seguro. Por ejemplo, puede conceder a los usuarios de MRS únicamente los permisos para realizar operaciones especificadas en clústeres de MRS, como crear un clúster y consultar una lista de clústeres en lugar de eliminar un clúster. La mayoría de las políticas definen permisos basados en API. Para ver las acciones API admitidas por MRS, consulte **Políticas de permisos y acciones admitidas**.

Tabla 11-1 enumera todas las políticas de sistema admitidas por MRS.

Tabla 11-1 Políticas del sistema de MRS

Política	Descripción	Tipo
MRS FullAccess	Permisos de administrador para MRS. Los usuarios con estos permisos pueden operar y usar todos los recursos MRS.	Fine-grained policy
MRS CommonOperations	Permisos de usuario comunes para MRS. Los usuarios con estos permisos pueden usar MRS pero no pueden agregar ni eliminar recursos.	Fine-grained policy
MRS ReadOnlyAccess	Permiso de sólo lectura para MRS. Los usuarios a los que se han concedido estos permisos sólo pueden ver los recursos de MRS.	Fine-grained policy
MRS Administrator	Permisos: ● Todas las operaciones en MRS ● Los usuarios con permisos de esta política también deben tener permisos de las políticas **Tenant Guest** y **Server Administrator**.	RBAC policy

Tabla 11-2 enumera las operaciones comunes soportadas por cada política definida por el sistema o función de MRS. Seleccione las políticas según sea necesario.

Tabla 11-2 Operaciones comunes apoyadas por cada política definida por el sistema

Operación	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Creación de un clúster	√	x	x	√

Operación	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Cambiar el tamaño de un clúster	√	x	x	√
Actualización de especificaciones de nodo	√	x	x	√
Eliminación de un clúster	√	x	x	√
Consulta de detalles del clúster	√	√	√	√
Consulta de una lista de clúster	√	√	√	√
Configuración de una regla de escalado automático	√	x	x	√
Consulta de una lista de host	√	√	√	√
Consulta de logs de operación	√	√	√	√
Creación y ejecución de un trabajo	√	√	x	√
Detener un trabajo	√	√	x	√
Supresión de un solo trabajo	√	√	x	√
Supresión de trabajos en lotes	√	√	x	√

Operación	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Consulta de detalles de trabajo	√	√	√	√
Consulta de una lista de trabajo	√	√	√	√
Creación de una carpeta	√	√	x	√
Eliminación de archivos	√	√	x	√
Consulta de una lista de archivo	√	√	√	√
Operación de etiquetas de clúster en lotes	√	√	x	√
Creación de una única etiqueta de clúster	√	√	x	√
Eliminación de una única etiqueta de clúster	√	√	x	√
Consulta de una lista de recursos por etiqueta	√	√	√	√
Consulta de etiquetas de clúster	√	√	√	√
Administrador de acceso	√	√	x	√
Consulta de una lista de parches	√	√	√	√
Instalación de un parche	√	√	x	√

Operación	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Desinstalación de un parche	√	√	x	√
Autorización de canales de O&M	√	√	x	√
Compartir logs de canales de O&M	√	√	x	√
Consulta de una lista de alarmas	√	√	√	√
Suscripción a la notificación de alarma	√	√	x	√
Envío de una sentencia de SQL	√	√	x	√
Consulta de resultados de SQL	√	√	x	√
Cancelación de una tarea de ejecución de SQL	√	√	x	√

MRS FullAccess

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "mrs:*:*",
                "ecs:*:*",
                "bms:*:*",
                "evs:*:*",
                "vpc:*:*",
                "bss:*:*",
                "kms:*:*",
                "rds:*:*"
            ],
```



```
            "Effect": "Allow"
        }
    ]
}
```

MRS CommonOperations

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "mrs:*:get*",
                "mrs:*:list*",
                "ecs:*:get*",
                "ecs:*:list*",
                "bms:*:get*",
                "bms:*:list*",
                "evs:*:get*",
                "evs:*:list*",
                "vpc:*:get*",
                "vpc:*:list*",
                "mrs:job:submit",
                "mrs:job:stop",
                "mrs:job:delete",
                "mrs:job:checkSql",
                "mrs:job:batchDelete",
                "mrs:file:create",
                "mrs:file:delete",
                "mrs:tag:batchOperate",
                "mrs:tag:create",
                "mrs:tag:delete",
                "mrs:manager:access",
                "mrs:patch:install",
                "mrs:patch:uninstall",
                "mrs:ops:grant",
                "mrs:ops:shareLog",
                "mrs:alarm:subscribe",
                "mrs:alarm:delete",
                "bss:*:get*",
                "bss:*:list*",
                "kms:*:get*",
                "kms:*:list*",
                "rds:*:get*",
                "rds:*:list*",
                "mrs:bootstrap:*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "mrs:cluster:create",
                "mrs:cluster:resize",
                "mrs:cluster:scaleUp",
                "mrs:cluster:delete",
                "mrs:cluster:policy"
            ],
            "Effect": "Deny"
        }
    ]
}
```


MRS ReadOnlyAccess

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Action": [
                "mrs:*:get*",
                "mrs:*:list*",
                "mrs:tag:count",
                "ecs:*:get*",
                "ecs:*:list*",
                "bms:*:get*",
                "bms:*:list*",
                "evs:*:get*",
                "evs:*:list*",
                "vpc:*:get*",
                "vpc:*:list*",
                "bss:*:get*",
                "bss:*:list*",
                "kms:*:get*",
                "kms:*:list*",
                "rds:*:get*",
                "rds:*:list*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "mrs:cluster:create",
                "mrs:cluster:resize",
                "mrs:cluster:scaleUp",
                "mrs:cluster:delete",
                "mrs:cluster:policy",
                "mrs:job:submit",
                "mrs:job:stop",
                "mrs:job:delete",
                "mrs:job:batchDelete",
                "mrs:file:create",
                "mrs:file:delete",
                "mrs:tag:batchOperate",
                "mrs:tag:create",
                "mrs:tag:delete",
                "mrs:manager:access",
                "mrs:patch:install",
                "mrs:patch:uninstall",
                "mrs:ops:grant",
                "mrs:ops:shareLog",
                "mrs:alarm:subscribe"
            ],
            "Effect": "Deny"
        }
    ]
}
```

MRS Administrator

```
{
    "Version": "1.0",
    "Statement": [
        {
            "Action": [
```



```
                "MRS:MRS:*"
            ],
            "Effect": "Allow"
        }
    ],
    "Depends": [
        {
            "catalog": "BASE",
            "display_name": "Server Administrator"
        },
        {
            "catalog": "BASE",
            "display_name": "Tenant Guest"
        }
    ]
}
```

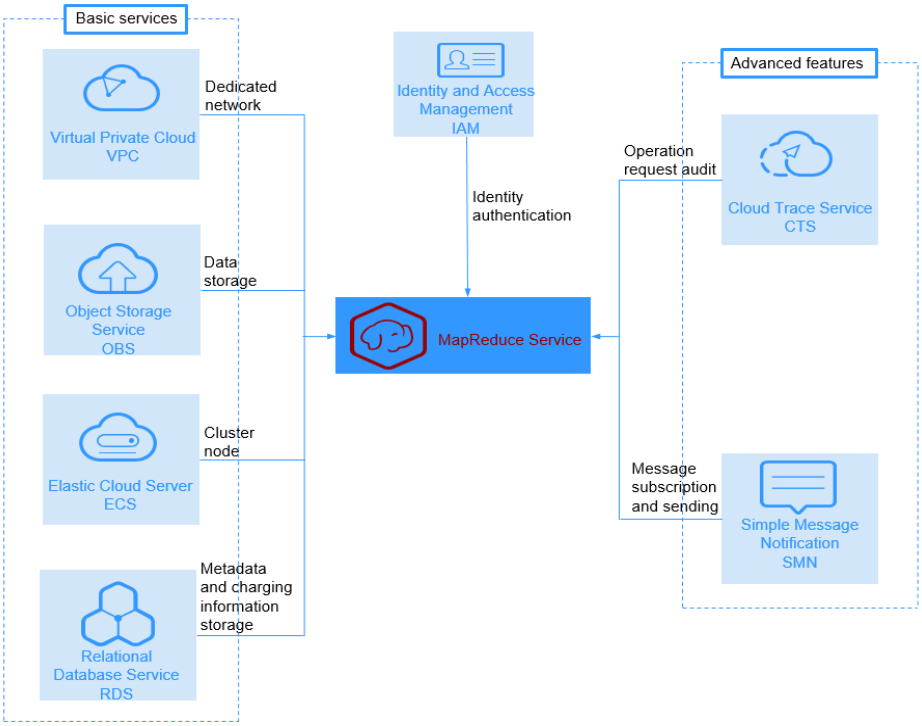
Enlaces útiles

- **Descripción del servicio de IAM**
- **Creación de grupos de usuarios y usuarios y concesión de permisos de MRS**
- **Políticas de permisos y acciones admitidas**

12 Servicios relacionados

Figura 12-1 muestra la relación entre MRS y otros servicios.

Figura 12-1 Relaciones con otros servicios



Relaciones con otros servicios

Tabla 12-1 Relaciones con otros servicios

Servicio	Relaciones	Referencia
Virtual Private Cloud (VPC)	Los clústeres de MRS se crean en las subredes de una VPC. Las VPC proporcionan un entorno de red seguro, aislado y lógico para sus clústeres de MRS.	**Creación de una VPC y una subred**

Servicio	Relaciones	Referencia
Object Storage Service (OBS)	OBS almacena los siguientes datos de usuario: ● Datos de entrada de trabajos de MRS, como programas de usuario y archivos de datos ● Datos de salida de trabajos de MRS, como archivos de resultados y archivos de registro de trabajos En clústeres de MRS, HDFS, Hive, MapReduce, YARN, Spark, Flume, y Loader pueden importar o exportar datos desde OBS. MRS utiliza el sistema de archivos paralelos de OBS para proporcionar servicios.	**Configuración de un clúster desacoplado del cálculo del almacenamiento (Agency)** **Configuración de un clúster desacoplado de cálculo de almacenamiento (AK/SK)**
Elastic Cloud Server (ECS)	MRS utiliza elastic cloud servers (ECSs) como nodos de clúster.	**Preparación de un entorno operativo** **Creación de un clúster**
Relational Database Service (RDS)	RDS almacena datos en ejecución del sistema MRS, incluido metadatos del clúster MRS e información de facturación del usuario.	**Configuración de conexiones de datos**
Identity and Access Management (IAM)	IAM proporciona autenticación para MRS.	**Creación de un usuario y concesión de permisos** **Creación de políticas personalizadas de MRS** **Sincronización de usuarios de IAM a MRS**
Simple Message Notification (SMN)	MRS utiliza SMN para proporcionar una suscripción y notificación de mensaje de uno a varios a través de una variedad de protocolos.	**Configuración de reglas de notificación de trabajos**
Cloud Trace Service (CTS)	CTS le proporciona registros de operaciones de solicitudes de operaciones de recursos MRS y resultados de solicitudes para consultas, auditorías y seguimiento.	**Tabla 12-2**

Tabla 12-2 Operaciones de MRS registradas por CTS

Operación	Tipo de recurso	Nombre del rastro
Creación de un clúster	cluster_mrs	createCluster
Eliminación de un clúster	cluster_mrs	deleteCluster
Expansión de un clúster	cluster_mrs	scaleOutCluster

Operación	Tipo de recurso	Nombre del rastro
Reducir un clúster	cluster_mrs	scaleInCluster

Después de habilitar CTS, el sistema inicia las operaciones de registro en los recursos de la nube. Puede ver los registros de operación de los últimos 7 días en la consola de gestión de CTS. Para obtener más información, consulte **Cloud Trace Service** >**Getting Started** >**Querying Real-Time Traces**.

13 Descripción de cuota

Las cuotas de recursos disponibles se configuran para cada cuenta de usuario en un entorno para evitar el abuso de recursos.

A continuación se enumeran las infraestructuras utilizadas por MapReduce. Las cuotas son gestionadas por cada servicio básico. Si necesita aumentar las cuotas, póngase en contacto con el soporte técnico del servicio correspondiente.

- ECS
- BMS
- VPC
- EVS
- Image Management Service (IMS)
- OBS
- EIP
- SMN
- IAM

Para obtener más información acerca de cómo ver y modificar las cuotas, consulte **Cuotas**.

14 Conceptos comunes

HBase Table

Una tabla HBase es un mapa tridimensional que comprende una o más columnas o filas de datos.

Columna

Columna es una dimensión de una tabla HBase. El nombre de la columna tiene el formato <*family*>**:**<*label*> donde <*family*> y <*label*> pueden ser cualquier combinación de caracteres. Una tabla HBase consta de un conjunto de familias de columnas. Cada columna de la tabla HBase pertenece a una familia de columnas.

Familia de columnas

Una familia de columnas es una colección de columnas almacenadas en el esquema HBase. Para crear columnas, primero debe crear una familia de columnas. Una familia de columnas organiza los datos con la misma propiedad en HBase. Cada fila de datos de la misma familia de columnas se almacena en el mismo servidor. Cada familia de columnas puede ser un atributo, como paquetes comprimidos, marcas de tiempo y caché de bloques de datos.

MemStore

MemStore es un núcleo del almacenamiento de HBase. Cuando la cantidad de datos almacenados en WAL alcanza el límite superior, los datos se cargan a MemStore para su clasificación y almacenamiento.

RegionServer

RegionServer es un servicio que se ejecuta en cada DataNode en el clúster de HBase. Es responsable de servir y gestionar las regiones, cargar la información de carga de las regiones y gestionar los nodos maestros distribuidos.

Marca de tiempo

Una marca de tiempo es un entero de 64 bits utilizado para indexar diferentes versiones de los mismos datos. Una marca de tiempo puede ser asignada automáticamente por HBase cuando los datos son escritos o asignados por los usuarios.

Store

Store es un núcleo del almacenamiento de HBase. Un Store aloja un MemStore y varios StoreFiles. Un Store corresponde a una familia de columnas de una tabla de una región.

Índice

Un índice es una estructura de datos que mejora la eficiencia de la recuperación de datos en una tabla de base de datos. Una o más columnas en una tabla de base de datos se pueden utilizar para la recuperación aleatoria rápida de datos y el acceso eficiente a los registros ordenados.

Coprocesador

Un coprocesador es una interfaz proporcionada por HBase para implementar la lógica de cálculo en RegionServer. Los coprocesadores se clasifican en coprocesadores de sistema y coprocesadores de tabla. El primero puede importar todas las tablas de datos de RegionServer y el segundo puede procesar una tabla especificada.

Block Pool

Un block pool es una colección de bloques que pertenecen a un solo espacio de nombres. DataNodes almacena bloques de todos los grupos de bloques de un clúster. Cada grupo de bloques se gestiona de forma independiente, lo que permite que un espacio de nombres genere un ID para un nuevo bloque sin depender de otros espacios de nombres. Si un NameNode no es válido, el DataNode todavía puede proporcionar servicios para otros NameNodes en el clúster.

DataNode

Un DataNode es un nodo de trabajo en el clúster HDFS. Programado por el cliente o NameNode, DataNodes almacena y recupera datos y periódicamente reporta bloques de archivos a NameNodes.

Bloque de archivo

Un bloque de archivo es la unidad lógica mínima almacenada en el HDFS. Cada archivo HDFS se almacena en uno o más bloques de archivo. DataNodes almacena todos los bloques de archivo.

Réplica de bloque

Una réplica es una copia de bloque almacenada en HDFS. Un bloque de archivo almacena varias réplicas para la disponibilidad del sistema y la tolerancia a fallos.

Namespace Volume

Un namespace volume es una unidad de gestión independiente que consta de un namespace y su block pool. Cuando se elimina un NameNode o namespace, también se eliminan los grupos de bloques relacionados en el DataNode. Durante una actualización de clúster, cada volumen de espacio de nombres se actualiza como un todo.

NodeManager

NodeManager ejecuta aplicaciones, supervisa el uso de recursos (incluidos CPU, memoria, discos y recursos de red) de las aplicaciones e informa el uso de recursos al ResourceManager.

ResourceManager

ResourceManager programa los recursos requeridos por las aplicaciones. Proporciona un complemento de programación para asignar recursos de clúster a múltiples colas y aplicaciones. El complemento de programación programa los recursos en función de las capacidades existentes o utilizando el modelo de programación justa.

Partición

Cada tema se puede dividir en varias particiones. Cada partición corresponde a un archivo de log adjunto cuya secuencia es fija.

Seguidor

Un seguidor procesa las solicitudes de lectura y trabaja con un líder para procesar las solicitudes de escritura. También se puede utilizar como una copia de respaldo de líder. Cuando el líder es defectuoso, un seguidor es elegido para hacerse cargo de la carga de trabajo del líder para evitar un único punto de falla.

Observador

Los observadores no participan en la votación para las elecciones y escriben solicitudes. Solo procesan solicitudes de lectura y reenvían solicitudes de escritura al líder, mejorando la eficiencia del procesamiento.

Líder

Un líder de los grupos ZooKeeper es elegido por los seguidores usando el protocolo de Zookeeper Atomic Broadcast (ZAB). Recibe y programa todas las solicitudes de escritura y sincroniza la información escrita con seguidores y observadores.

CarbonData

A Carbon es una arquitectura abierta basada en Spark SQL. Integra el motor MOLAP desarrollado por Huawei y Spark, y construye rápidamente el motor de análisis multidimensional distribuido basado en Spark, acortando la duración del análisis de minutos a segundos y reforzando la capacidad de análisis multidimensional de Spark.

DStream

DStream es un concepto abstracto proporcionado por Spark Streaming. Es un flujo de datos continuo que se obtiene de la fuente de datos o el flujo de entrada transformado. En esencia, un DStream es una serie de conjuntos de datos distribuidos resilientes continuos (RDD).

Memoria en el montón

Un montón indica el área de datos donde se está ejecutando la máquina virtual de Java (JVM) y desde la que se ha comprometido la memoria para todas las instancias de clase y matrices.

Los parámetros de inicio de JVM **-Xms** y **-Xmx** se utilizan para establecer la memoria en el montón inicial y la memoria en el montón máxima, respectivamente.

- Memoria en el montón máxima: memoria en el montón que el sistema puede asignar a un programa como máximo, especificada por el parámetro **-Xmx**.
- Memoria en el montón asignada: memoria en el montón total asignada por el sistema para ejecutar un programa. Se extiende desde la memoria en el montón inicial y la memoria en el montón máxima.
- Memoria en el montón usada: memoria en el montón usada por un programa. Es más pequeño que la memoria en el montón asignada.
- Memoria no acumulativa: memoria excluida de los montones de JVM y del área de memoria para ejecutar la JVM. La memoria no acumulativa tiene los tres grupos de memoria siguientes:
  - Caché de código: almacena el código compilado de JIT. Su valor se establece a través del parámetro de inicio de JVM **-XX:InitialCodeCacheSize -XX:ReservedCodeCacheSize**. El valor predeterminado es 240 MB.
  - Espacio de clase comprimido: almacena metadatos de un puntero. Su valor se establece a través del parámetro de inicio de JVM **-XX:CompressedClassSpaceSize**. El valor predeterminado es 1024 MB.
  - Metaspace: almacena metadatos. Su valor se establece a través del parámetro de inicio de JVM **-XX:MetaspaceSize -XX:MaxMetaspaceSize**.
- Máxima memoria no acumulativa: memoria no acumulativa asignada a un programa como máximo por el sistema. Su valor es la suma de los valores máximos de Code Cache, Compressed Class Space y Metaspace.
- Memoria no acumulativa asignada: memoria no acumulativa total asignada por el sistema para ejecutar un programa. Se extiende desde la memoria inicial no acumulativa y la memoria máxima no acumulativa.
- Memoria no acumulativa usada: memoria no acumulativa que ha sido usada por aprogram. Es más pequeño que la memoria no acumulada.

Hadoop

Hadoop es un marco de sistema distribuido. Permite a los usuarios desarrollar aplicaciones distribuidas utilizando computación de alta velocidad y almacenamiento proporcionado por clústeres sin conocer los detalles subyacentes del sistema distribuido. También puede procesar de manera fiable y eficiente cantidades masivas de datos en modo escalable y distribuido. Hadoop es confiable porque mantiene múltiples duplicados de datos de trabajo, lo que permite el procesamiento distribuido para nodos con errores. Hadoop es altamente eficiente porque procesa datos en modo paralelo. Hadoop es escalable porque puede procesar petabytes de datos. Hadoop está compuesto por HDFS, MapReduce, HBase, y Hive.

Rol

Un rol es un elemento de un servicio. Un servicio contiene uno o varios roles. Los servicios se instalan en los servidores a través de roles para que puedan ejecutarse correctamente.

Clúster

Un clúster es una tecnología informática que permite que varios servidores funcionen como un solo servidor. Los clústeres mejoran la estabilidad, la confiabilidad y la capacidad de procesamiento o servicio de datos del sistema. Por ejemplo, los clústeres pueden evitar fallos

de punto único (SPOF), compartir recursos de almacenamiento, reducir la carga del sistema y mejorar el rendimiento del sistema.

instancia

Se forma una instancia cuando se instala un rol de servicio en el host. Un servicio tiene una o más instancias de rol.

Metadatos

Los metadatos son datos que proporcionan información sobre otros datos y también se denominan datos de medios o datos de retransmisión. Se utiliza para definir propiedades de datos, especificar ubicaciones de almacenamiento de datos y datos históricos, recuperar recursos y archivos de registro.

