MapReduce Service

Visão geral de serviço

Edição 01

Data 14-08-2023

HUAWEI TECHNOLOGIES CO., LTD.

Marcas registadas e permissões


e outras marcas registadas da Huawei são marcas registadas da Huawei Technologies Co., Ltd. Todos as outras marcas registadas e os nomes registados mencionados neste documento são propriedade dos seus respectivos detentores.

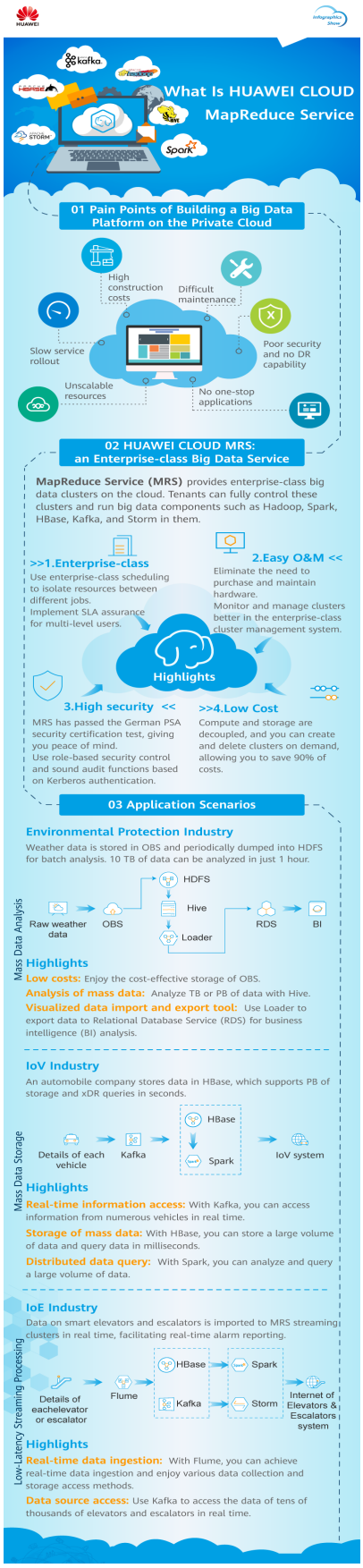
Aviso

Os produtos, serviços e funcionalidades adquiridos são estipulados pelo contrato feito entre a Huawei e o cliente. Todos ou parte dos produtos, serviços e funcionalidades descritos neste documento pode não estar dentro do âmbito de aquisição ou do âmbito de uso. Salvo especificação em contrário no contrato, todas as declarações, informações e recomendações neste documento são fornecidas "TAL COMO ESTÁ" sem garantias, ou representações de qualquer tipo, seja expressa ou implícita.

As informações contidas neste documento estão sujeitas a alterações sem aviso prévio. Foram feitos todos os esforços na preparação deste documento para assegurar a exatidão do conteúdo, mas todas as declarações, informações e recomendações contidas neste documento não constituem uma garantia de qualquer tipo, expressa ou implícita.

Índice

1 Infográficos



2 O que é o MRS?

Big Data é um enorme desafio enfrentado pela era da Internet, à medida que o volume e os tipos de dados aumentam rapidamente. As tecnologias convencionais de processamento de dados, como armazenamento de nó único e bancos de dados relacionais, não conseguem resolver os problemas emergentes de Big Data. Nesse caso, a Apache Software Foundation (ASF) lançou uma solução de processamento de Big Data de Hadoop de código aberto. O Hadoop é uma plataforma de computação distribuída de código aberto que pode utilizar totalmente os recursos de computação e armazenamento de clusters para processar grandes quantidades de dados. Se as empresas implementarem sistemas Hadoop sozinhas, as desvantagens incluem altos custos, longo período de implementação, manutenção difícil e uso inflexível.

Para resolver esses problemas, o Serviço MapReduce (MRS) é fornecido na Huawei Cloud para você gerenciar componentes baseados em Hadoop. Com o MRS, você pode implementar um cluster do Hadoop com apenas alguns cliques. MRS fornece clusters de Big Data na nuvem para empresas. Os locatários podem controlar totalmente os clusters e executar facilmente componentes de Big Data, como Storm, Hadoop, Spark, HBase e Kafka. MRS é inteiramente compatível com APIs de código aberto, além de incorporar as vantagens da computação da Huawei Cloud e do armazenamento e da experiência na indústria de Big Data para fornecer uma plataforma de Big Data de pilha completa de alto desempenho, baixo custo, flexibilidade e facilidade de uso. Além disso, a plataforma pode ser personalizada com base nos requisitos de serviço para ajudar as empresas a construir rapidamente um sistema de processamento de dados massivo e descobrir novos pontos de valor e oportunidades de negócios, analisando e minerando grandes quantidades de dados em tempo real ou em tempo não real.

Arquitetura do produto

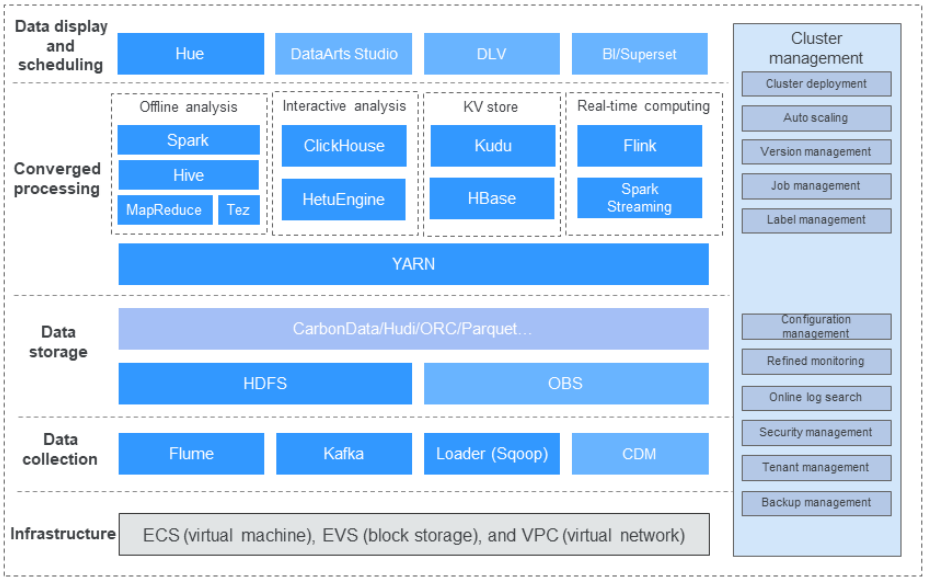
Lista de versões de componentes do MRS lista as versões dos componentes do MRS.

Figura 2-1 mostra a arquitetura lógica do MRS.

NOTA

O MRS 3.x ou posterior não oferece suporte ao gerenciamento de patches no console de gerenciamento.

Figura 2-1 Arquitetura do MRS



A arquitetura do MRS inclui fases de infraestrutura e processamento de Big Data.

- Infraestrutura

  Os clusters de Big Data da MRS são construídos com base no Elastic Cloud Server (ECS) da Huawei Cloud e utilizam totalmente os recursos de alta confiabilidade e segurança da camada de virtualização.

  - Uma Virtual Private Cloud (VPC) é uma rede interna virtual fornecida para cada locatário. Ela é isolada de outras redes por padrão.
  - O Elastic Volume Service (EVS) fornece armazenamento altamente confiável e de alto desempenho.
  - O ECS fornece VMs escaláveis e funciona com VPCs, grupos de segurança e o mecanismo de múltiplas réplicas do EVS para criar um ambiente de computação eficiente, confiável e seguro.

- Coleta de dados

  A camada de coleta de dados fornece a capacidade de importar dados de várias fontes dta, como Flume (ingestão de dados), Loader (importação de dados relacionais) e Kafka (fila de mensagens altamente confiável), para clusters de Big Data do MRS. Como alternativa, você pode usar a Cloud Data Migration (CDM) para importar dados externos para clusters do MRS.

- Armazenamento de dados

  Os clusters de MRS podem armazenar dados estruturados e não estruturados e oferecer suporte a vários formatos eficientes para atender aos requisitos de diferentes mecanismos de computação.

  - O HDFS é um sistema de arquivos distribuído de uso geral em uma plataforma de Big Data.
  - O OBS é um serviço de armazenamento de objetos que apresenta alta disponibilidade e baixo custo.
  - O HBase suporta armazenamento de dados com índices e é aplicável a cenários de consulta baseados em índices de alto desempenho.

- Processamento de convergência de dados
  - O MRS fornece vários mecanismos de computação principais, incluindo MapReduce (processamento em lote), Tez (modelo DAG), Spark (computação em memória), Storm (computação de fluxo) e Flink (computação de fluxo) para converter estruturas de dados e lógica em modelos de dados que atendem aos requisitos de serviço em uma variedade de cenários de aplicações de Big Data.
  - Com base no modelo de dados predefinido e na análise de dados de SQL fácil de usar, os usuários podem selecionar Hive (data warehouse), SparkSQL e Presto (mecanismo de consulta interativo).
- Exibição e agendamento de dados

  Exibe resultados de análise de dados e integra-se ao DataArts Studio para fornecer uma plataforma de desenvolvimento colaborativo de Big Data completa, ajudando você a concluir várias tarefas, como modelagem de dados, integração de dados, desenvolvimento de scripts, agendamento de jobs e monitoramento de O&M. e ajudá-lo a construir centros de processamento de Big Data sem esforço.
- Gerenciamento de cluster

  Todos os componentes do ecossistema de Big Data baseado em Hadoop são implementados em modo distribuído, e sua implementação, gerenciamento e O&M são complexos.

  O MRS fornece uma plataforma unificada de gerenciamento de O&M para gerenciamento de clusters, com suporte para implementação de clusters com um clique, seleção de várias versões, bem como dimensionamento manual e dimensionamento automático de clusters sem interrupção de serviço. Além disso, o MRS fornece gerenciamento de tarefas, gerenciamento de tags de recursos e O&M dos componentes de processamento de dados anteriores em cada camada. Ele também fornece recursos de O&M completos, abrangendo monitoramento, relatórios de alarmes, configuração e atualização de patches.

Vantagens do produto

O MRS tem uma poderosa equipe de kernel Hadoop e é implementada com base na plataforma de Big Data FusionInsight de nível empresarial da Huawei. O MRS foi implementado em dezenas de milhares de nós e pode garantir acordos de nível de serviço (SLAs) para usuários de vários níveis.

MRS possui as seguintes vantagens:

- Alto desempenho

  O MRS oferece suporte à tecnologia de armazenamento CarbonData autodesenvolvido. O CarbonData é uma solução de armazenamento de Big Data de alto desempenho. Ele permite que um conjunto de dados seja aplicado a vários cenários e oferece suporte a recursos como indexação de vários níveis, codificação de dicionário, pré-agregação, particionamento dinâmico e consulta de dados em tempo quase real. Isso melhora a digitalização de I/O e o desempenho de computação e retorna resultados de análise de dezenas de bilhões de registros de dados em segundos. Além disso, o MRS suporta o agendador aprimorado autodesenvolvido Superior, que quebra o gargalo de escala de um único cluster e é capaz de agendar por mais de 10.000 nós em um cluster.
- Econômico

  Com base em infraestrutura de nuvem diversificada, o MRS oferece várias opções de computação e armazenamento e separa a computação do armazenamento, oferecendo soluções econômicas de armazenamento de dados em massa. O MRS oferece suporte ao

dimensionamento automático para lidar com cargas de serviço de pico e fora de pico, liberando recursos ociosos na plataforma de Big Data para os clientes. Os clusters de MRS podem ser criados e dimensionados quando você precisar deles e podem ser encerrados ou dimensionados depois de usá-los, minimizando o custo.

- Alta segurança

  O MRS fornece gerenciamento de permissões multi-locatário de Big Data de nível empresarial e gerenciamento de segurança para suportar controle de acesso baseado em tabela e coluna e criptografia de dados.

- O&M fácil

  O MRS fornece uma plataforma de gerenciamento de cluster de Big Data visualizada, melhorando a eficiência de O&M. O MRS suporta atualização contínua de patch e fornece informações visualizadas sobre o lançamento de patch e instalação de patch com um clique sem intervenção manual, garantindo a estabilidade a longo prazo dos clusters de usuários.

- Alta confiabilidade

  A confiabilidade em larga escala comprovada e a estabilidade a longo prazo do MRS atendem aos requisitos de alta confiabilidade de nível empresarial. Além disso, o MRS suporta backup automático de dados em AZs e regiões, bem como antiafinidade automática. Ele permite que as VMs sejam distribuídas em diferentes máquinas físicas.

Usar o MRS pela primeira vez

Se você é um usuário iniciante, familiarize-se com as seguintes informações:

- **Conceitos básicos**

  Consulte **Componentes** e **Funções** para aprender o conhecimento básico do MRS, incluindo os princípios básicos e os recursos aprimorados de cada componente do MRS, bem como os conceitos e funções exclusivos do MRS.

- **Primeiros passos**

  Para saber como usar o MRS, consulte **Primeiros passos do MapReduce Service**. "Primeiros passos" fornece a orientação detalhada da operação das amostras. Você pode criar e usar clusters de MRS com base nas orientações de operação.

- **Outras funções e guias de operação**

  Se você for um usuário de cluster do MRS e engenheiro de O&M, poderá executar operações como gerenciamento do ciclo de vida do cluster, escalabilidade e gerenciamento de jobs consultando **Guia de usuário do MapReduce Service**. Consulte **Guia de operação de componente do MapReduce Service** para saber como usar componentes em um cluster.

  Se você é um desenvolvedor, pode consultar o guia de operação e exemplos de projetos no **Guia de desenvolvimento do MapReduce Service** do MRS para desenvolver, executar e comissionar suas próprias aplicações. Você também pode chamar APIs para gerenciar clusters do MRS e executar jobs. Para obter detalhes, consulte **Referência de API do MapReduce Service**.

3 Vantagens do MRS em comparação ao Hadoop autoconstruído

MRS fornece clusters de Big Data na nuvem para empresas. Os locatários podem controlar totalmente os clusters e executar componentes de Big Data como Hadoop, Spark, HBase, Kafka e Storm com facilidade. O MRS libera você da compra e manutenção de hardware. MRS é desenvolvido com base na plataforma FusionInsight de Big Data de classe empresarial da Huawei e foi implementado em dezenas de milhares de nós do setor, fornecendo garantia multinível de SLA com suporte profissional aos serviços de kernel de Hadoop. Quando comparado aos clusters de Hadoop desenvolvidos internamente, MRS tem as seguintes vantagens:

1. **MRS oferece suporte a criação, exclusão e dimensionamento de cluster com um clique. Você pode utilizar um endereço IP elástico (EIP) para acessar o MRS Manager, facilitando o uso de clusters de Big Data.**
   - Clusters de Big Data auto-construídos apresentam problemas como altos custos, longos períodos, O&M difíceis e inflexíveis. Para resolver esses problemas, o MRS fornece criação, exclusão, dimensionamento e dimensionamento de cluster com um clique, permitindo que você personalize o tipo de cluster, o intervalo de componentes, o número de nós de cada tipo, as especificações da VM, as zonas de disponibilidade (AZs), a rede da VPC e informações de autenticação. O MRS pode criar automaticamente um cluster que atenda aos requisitos de configuração. Além disso, você pode criar rapidamente clusters de vários aplicativos, por exemplo, cluster de análise de Hadoop, cluster de HBase e cluster de Kafka. O MRS oferece suporte à implementação de cluster heterogêneo. Ou seja, VMs de especificações diferentes podem ser combinadas em um cluster com base em tipos de CPU, capacidades de disco, tipos de disco e tamanhos de memória.
   - O MRS fornece um canal seguro baseado em EIP para você acessar facilmente as IUs da Web dos componentes. Isso é mais conveniente do que vincular um EIP por conta própria, e você pode acessar as IUs da Web com alguns cliques, evitando as etapas de logon em uma VPC, adicionando regras de grupo de segurança e obtendo um endereço IP público.
   - O MRS fornece ações de inicialização personalizadas para configurar de forma flexível seus clusters dedicados. O software de terceiros que não é suportado pelo MRS pode ser instalado automaticamente, permitindo que você execute operações personalizadas, como modificar o ambiente em execução do cluster.
   - O MRS suporta o recurso WrapperFS, fornece o recurso de tradução do OBS (ou seja, acesso ao OBS por meio do mapeamento de endereços) e pode migrar dados

sem problemas do HDFS para o OBS. Após a migração, você pode acessar os dados armazenados no OBS de clientes sem modificar a lógica do código de serviço.

2. **MRS possibilita o dimensionamento automático, que é mais econômico do que o cluster de Hadoop autoconstruído.**

   O MRS suporta o escalonamento automático para endereçar cargas de serviço de pico e fora de pico. Ele se aplica a recursos extras durante as horas de pico e libera recursos ociosos durante as horas de pico, ajudando você a economizar recursos ociosos na plataforma de Big Data durante as horas de pico, minimizar custos e se concentrar nos serviços principais.

   Em aplicações de Big Data, especialmente em análise e processamento periódicos de dados, os recursos de computação de cluster precisam ser ajustados dinamicamente com base nas alterações de dados de serviço para atender aos requisitos de serviço. A função de escalonamento automático do MRS permite que os clusters sejam escalonados elasticamente ou in com base nas cargas do cluster. Além disso, se o volume de dados mudar regularmente e você quiser escalar para fora ou em um cluster antes das alterações de volume de dados, você pode usar o recurso de plano de recursos do MRS. O MRS suporta dois tipos de políticas de escalonamento automático: regras de escalonamento automático e planos de recursos

   - Regras de dimensionamento automático: você pode aumentar ou diminuir os nós de Tarefa com base nas cargas de cluster em tempo real. O dimensionamento automático será acionado quando o volume de dados mudar, mas pode haver algum atraso.
   - Planos de recursos: se o volume de dados mudar periodicamente, você poderá criar planos de recursos para redimensionar o cluster antes que o volume de dados seja alterado, evitando assim um atraso no aumento ou diminuição de recursos.

   Tanto as regras de dimensionamento automático quanto os planos de recursos podem acionar o dimensionamento automático. Você pode configurar ambos ou configurar um deles. A configuração de planos de recursos e regras de dimensionamento automático melhora a escalabilidade do nó do cluster para lidar com picos de volume de dados ocasionalmente inesperados.

3. **MRS possibilita o desacoplamento entre armazenamento e computação, melhorando muito a utilização de recursos de clusters de Big Data.**

   Na arquitetura tradicional de Big Data, onde os recursos de armazenamento e computação são integrados, a expansão é difícil e os recursos não são bem utilizados. Para resolver esses problemas, o MRS adota uma arquitetura de separação de armazenamento de computação. Com base no OBS, o armazenamento atinge 99,999999999% de confiabilidade e capacidade ilimitada, suportando o crescimento contínuo de dados corporativos. Os recursos de computação podem ser dimensionados elasticamente dentro ou fora de 0 a *N* nós. Centenas de nós podem ser provisionados rapidamente. Com a nova arquitetura, os nós de computação podem ser escalonados elasticamente. O armazenamento de dados entre AZs baseado em OBS garante maior confiabilidade, livra você de se preocupar com emergências, como terremotos e cortes de fibra. Os recursos de armazenamento e computação podem ser configurados de forma flexível e dimensionados de forma elástica, conforme necessário. Isso torna a alocação de recursos mais precisa e razoável, melhorando significativamente a utilização de recursos de clusters de Big Data e reduzindo o custo de análise abrangente em 50%.

   Além disso, a arquitetura de separação de computação-armazenamento de alto desempenho rompe o limite da computação paralela da arquitetura de armazenamento-computação integrada. Ele maximiza a alta largura de banda e a alta simultaneidade do OBS e otimiza a eficiência de acesso a dados e a computação paralela aprofundada

(como operação de metadados e otimização de algoritmo de gravação) para melhorar o desempenho.

4. **MRS oferece suporte ao CarbonData e ao Superior Scheduler desenvolvidos pela Huawei, proporcionando um melhor desempenho.**
   - O MRS oferece suporte à tecnologia de armazenamento CarbonData desenvolvida por você mesmo. O CarbonData é uma solução de armazenamento de Big Data de alto desempenho. Ele permite que um conjunto de dados seja aplicado a vários cenários e oferece suporte a recursos como indexação de vários níveis, codificação de dicionário, pré-agregação, particionamento dinâmico e consulta de dados em tempo quase real. Isso melhora a digitalização de I/O e o desempenho de computação e retorna resultados de análise de dezenas de bilhões de registros de dados em segundos.
   - Além disso, o MRS oferece suporte ao Superior Scheduler autodesenvolvido, que aumenta a capacidade de escala de um único cluster e é capaz de agendar maid de 10.000 nós em um cluster. O Superior Scheduler é um mecanismo de agendamento projetado para o sistema de gerenciamento de recursos distribuídos de Hadoop YARN. É um programador de alto desempenho e de nível empresarial projetado para pools de recursos convergentes e requisitos de serviço multi locatário. O Superior Scheduler atinge todas as funções dos agendadores de código aberto, do Fair Scheduler e do Capacity Scheduler. Em comparação com os programadores de código aberto, o Superior Scheduler é aprimorado na política de agendamento de recursos multi locatário da empresa, isolamento e compartilhamento de recursos por vários usuários em um locatário, desempenho de agendamento, utilização de recursos do sistema e escalabilidade de cluster, e foi projetado para substituir os programadores de código aberto.
5. **MRS otimiza software e hardware baseados em processadores Kunpeng para liberar totalmente a capacidade computacional do hardware e alcançar o custo-benefício necessário.**

   O MRS suporta servidores Kunpeng autodesenvolvidos, cujos recursos multi-núcleo e de alta concorrência são totalmente utilizados para fornecer chips auto-otimizados de pilha completa e usa EulerOS, Huawei JDK e camada de aceleração de dados para garantir o desempenho do hardware, proporcionando alto poder de computação para computação de Big Data. Com o desempenho semelhante, o custo da solução de Big Data de ponta a ponta é reduzido em 30%.
6. **MRS suporta diversos modos de isolamento e gerenciamento de permissão de múltiplos locatários de Big Data empresarial, garantindo maior segurança.**
   - O MRS suporta a implantação de recursos e o isolamento de recursos físicos em zonas dedicadas. Você pode combinar recursos de computação e armazenamento de forma flexível, como recursos de computação dedicados + recursos de armazenamento compartilhado, recursos de computação compartilhados + recursos de armazenamento dedicados e recursos de computação dedicados + recursos de armazenamento dedicados. Um cluster de MRS suporta vários locatários lógicos. O isolamento de permissões permite que os recursos de computação, armazenamento e tabela do cluster sejam divididos com base nos locatários.
   - Com a autenticação Kerberos, o MRS fornece controle de acesso baseado em função (RBAC) e funções de auditoria de som.
   - Com o Cloud Trace Service (CTS) sendo interconectado com o MRS, você recebe registros de operação de solicitações de operação de recursos do MRS e resultados de solicitações para consulta, auditoria e rastreamento inverso. Você pode usar o CTS para auditar e rastrear todas as operações de cluster.

- Está provado que, com o Host Security Service (HSS) interconectado com o MRS, a segurança do serviço é aprimorada sem deteriorar as funções e o desempenho.
- O MRS oferece suporte ao fazer logon de usuário unificado com base na interface do usuário da Web. O Manager fornece autenticação de usuário, que lhe concede permissão para acessar um cluster.
- O MRS suporta criptografia de armazenamento de dados, armazenamento criptografado de todas as contas de usuário e senhas, transmissão criptografada de canais de dados e autenticação de certificado bidirecional para acesso a dados entre zonas confiáveis de módulos de serviço.
- Os clusters de Big Data do MRS fornecem uma solução multi locatário completa para Big Data de nível empresarial. Multi-locatário refere-se a uma coleção de vários recursos (cada conjunto de recursos é um locatário) em um cluster de Big Data do MRS. Ele pode alocar e agendar recursos, incluindo recursos de computação e armazenamento. Multi-locatário isola os recursos de um cluster de Big Data em conjuntos de recursos. Os usuários podem alugar conjuntos de recursos desejados para executar aplicações e jobs e armazenar dados. Em um cluster de Big Data, vários conjuntos de recursos podem ser implementados para atender a diversos requisitos de vários usuários.
- O MRS oferece suporte ao gerenciamento de permissões refinado. Com o recurso de autorização refinado fornecido pelo IAM da HUAWEI CLOUD, o MRS pode especificar as operações, os recursos e as condições de solicitação de serviços específicos. Esse mecanismo permite uma autorização baseada em políticas mais flexível, atendendo aos requisitos de controle de acesso seguro. Por exemplo, você pode conceder aos usuários do MRS somente as permissões para executar operações especificadas em clusters do MRS, como criar um cluster e consultar uma lista de clusters em vez de excluir um cluster. Além disso, o MRS suporta o gerenciamento de permissões refinado do OBS para vários locatários. As permissões para acessar buckets e objetos do OBS nos buckets são diferenciadas com base nas funções do usuário, para que os usuários do MRS possam controlar um diretório diferente nos buckets do OBS.
- O MRS suporta o gerenciamento de projetos empresariais. O projeto empresarial é uma maneira de gerenciar os recursos da nuvem. O Enterprise Management fornece serviços de gerenciamento abrangentes para clientes empresariais, como recursos de nuvem, pessoal, permissões e status financeiro. Os consoles de gerenciamento comuns são orientados para o controle e a configuração de produtos individuais de nuvem. O console do Enterprise Management, em contraste, é mais focado no gerenciamento de recursos. Ele é projetado para ajudar as empresas a gerenciar recursos, pessoal, permissões e finanças baseados em nuvem, de maneira hierárquica, como gerenciamento de empresas, departamentos e projetos. O MRS permite que os usuários que ativaram o Enterprise Project Management Service (EPS) para configurar projetos empresariais para um cluster durante a criação do cluster e usem o EPS para gerenciar recursos do MRS por grupo. Esse recurso é aplicável a cenários em que você precisa gerenciar vários recursos por grupo e executar operações como controle de permissão e consulta de taxa baseada em projeto em projetos corporativos.

7. **MRS implementa a alta disponibilidade para todos os nós de gerenciamento e oferece suporte a um mecanismo de confiabilidade abrangente, tornando o sistema mais confiável.**

   Baseado no software de código aberto Apache Hadoop, o MRS otimiza e melhora a confiabilidade dos principais componentes do serviço.

- HA para todos os nós de gerenciamento

  Na versão de código aberto do Hadoop, os dados e os nós de computação são gerenciados em um sistema distribuído, no qual um único ponto de falha (SPOF) não afeta a operação de todo o sistema. No entanto, um SPOF pode ocorrer em nós de gerenciamento em execução no modo centralizado, o que se torna o ponto fraco da confiabilidade geral do sistema.

  O MRS fornece mecanismos de nó duplo semelhantes para todos os nós de gerenciamento dos componentes do serviço, como Manager, Presto, HDFS NameNodes, Hive Servers, HBase HMasters, YARN Resource Managers, Kerberos Servers e Ldap Servers. Todos eles são implementados em modo ativo/em espera ou configurados com compartilhamento de carga, impedindo efetivamente que SPOFs afetem a confiabilidade do sistema.

- Mecanismo de confiabilidade abrangente

  Por análise de confiabilidade, as seguintes medidas para lidar com exceções de software e hardware são fornecidas para melhorar a confiabilidade do sistema:

  - Depois que a fonte de alimentação é restaurada, os serviços funcionam adequadamente, independentemente de uma falha de energia de um único nó ou de todo o cluster, garantindo a confiabilidade dos dados em caso de falhas de energia inesperadas. Os dados principais não serão perdidos a menos que o disco rígido esteja danificado.
  - As verificações de estado de integridade e o tratamento de falhas do disco rígido não afetam os serviços.
  - As falhas do sistema de arquivos podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.
  - As falhas de processo e nó podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.
  - As falhas de rede podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.

8. **MRS fornece uma interface de gerenciamento de cluster de Big Data visualizada de maneira unificada, facilitando o O&M.**

   - Na interface de gerenciamento do cluster de Big Data, a inicialização e a interrupção do serviço, a modificação da configuração e a verificação de integridade estão disponíveis. O MRS também fornece funções de gerenciamento, monitoramento e alarme de cluster visualizados e convenientes. Além disso, você pode verificar e auditar o status de integridade do sistema em um clique, garantindo a execução normal do sistema e reduzindo os custos de O&M do sistema.
   - Depois que a Simple Message Notification (SMN) é configurada, o MRS pode enviar informações de status de integridade do cluster em tempo real, incluindo alterações de cluster e alarmes de componentes em tempo real para você por meio de mensagens SMS ou e-mails, facilitando O&M, monitoramento em tempo real e envio de alarmes em tempo real.
   - O MRS oferece suporte à atualização contínua de patches e fornece informações visualizadas sobre a liberação de patches e a instalação de patches com um clique sem intervenção manual, garantindo a estabilidade a longo prazo dos clusters de usuários.
   - Se ocorrer um problema quando você usar um cluster de MRS, poderá iniciar a autorização de O&M no console de gerenciamento de MRS. A equipe de O&M pode ajudá-lo a localizar rapidamente o problema e você pode revogar a autorização a qualquer momento. Você também pode iniciar o compartilhamento de logs no

console de gerenciamento do MRS para compartilhar um escopo de log especificado com a equipe de O&M, de modo que a equipe de O&M possa localizar falhas sem acessar o cluster.

- O MRS suporta o despejo de logs sobre falhas de criação de cluster para o OBS para que o pessoal de O&M obtenha e analise os logs.

9. **MRS tem um ecossistema aberto e possibilita a interconexão sem falhas com serviços periféricos, permitindo que você construa rapidamente uma plataforma unificada de Big Data.**
   - Com base no MRS, um serviço de Big Data de pilha completa, as empresas podem criar uma plataforma unificada de Big Data para ingestão, armazenamento, análise e mineração de valor com um clique e interconectar-se com o DataArts Studio e os serviços de visualização de dados para ajudar os clientes a migrar facilmente dados para a nuvem, desenvolver e programar jobs de Big Data e exibir dados. Isso libera os clientes da construção complexa de plataformas de Big Data e da calibração e manutenção profissionais de Big Data, para que os clientes possam se manter mais focados em aplicações do setor e usar uma cópia de dados em vários cenários de serviços. O DataArts é uma plataforma de operações de desenvolvimento de ciclo de vida de dados que fornece uma ampla gama de funções, como integração de dados, desenvolvimento, governança, serviço e visualização. Os dados do DataArts podem ser ingeridos no Estúdio DataArts para desenvolvimento colaborativo visualizado com um clique, aproveitando a GUI visualizada do DataArts Studio, tipos de desenvolvimento de dados abundantes (script e job), agendamento de job totalmente hospedado e monitoramento de O&M. e pipelines de processamento de dados da indústria embutidos. Isso torna o Big Data muito mais fácil de usar, ajuda você a construir rapidamente centros de processamento de Big Data e permite a monetização rápida.
   - O MRS é totalmente compatível com o ecossistema de Big Data de código aberto. Com abundantes ferramentas de migração de dados e aplicações, o MRS ajuda você a migrar rapidamente dados de suas próprias plataformas sem modificação de código e interrupção de serviço.

4 Cenários de aplicação

Big Data é onipresente em nossas vidas. Huawei Cloud MRS é adequado para processar Big Data em setores como Internet das Coisas (IoT), comércio eletrônico, finanças, manufatura, saúde, energia e departamentos governamentais.

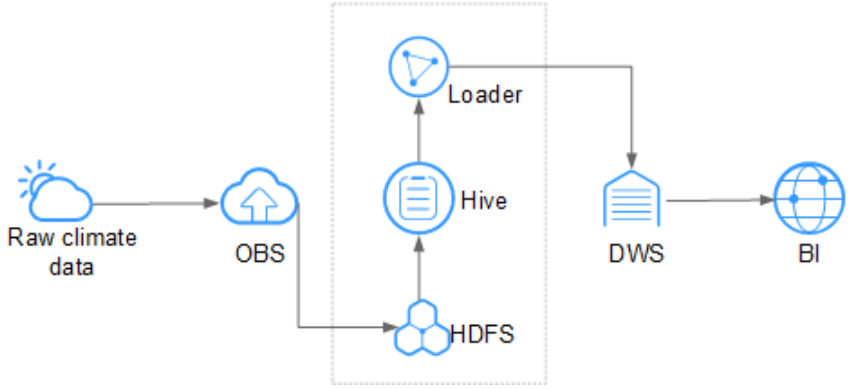
Análise de dados em larga escala

A análise de dados em larga escala é um cenário importante nos sistemas modernos de Big Data. Geralmente, uma empresa tem várias fontes de dados. Depois que os dados são acessados, o processamento de extração, transformação e carga (ETL) é necessário para gerar dados modelados para cada módulo de serviço para analisar e classificar os dados. Este tipo de serviço tem as seguintes características:

- Os requisitos para execução em tempo real não são altos, e o tempo de execução de job varia de dezenas de minutos a horas.
- O volume de dados é grande.
- Existem várias fontes de dados e formatos diversificados.
- O processamento de dados geralmente consiste em várias tarefas, e os recursos precisam ser planejados em detalhes.

Na indústria de proteção ambiental, os dados climáticos são armazenados no OBS e periodicamente despejados no HDFS para análise em lote. 10 TB de dados climáticos podem ser analisados em 1 hora.

Figura 4-1 Análise de dados em larga escala na indústria de proteção ambiental



O MRS tem as seguintes vantagens neste cenário.

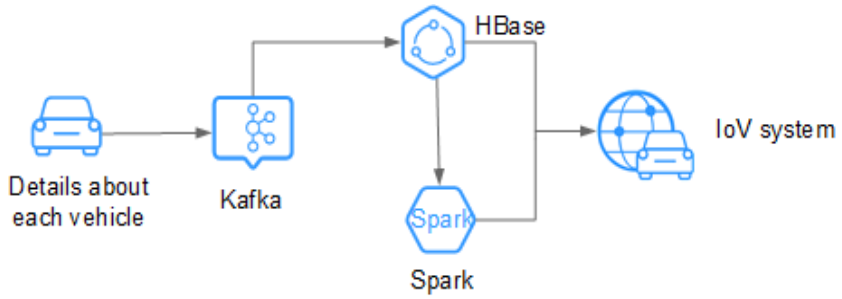
- Baixo custo: o OBS oferece armazenamento econômico.
- Análise maciça de dados: os dados de nível TB/PB são analisados pelo Hive.
- Ferramenta de importação e exportação de dados visualizados: o Loader exporta dados para o Data Warehouse Service (DWS) para análise de business intelligence (BI).

Armazenamento de dados em grande escala

Um usuário que tem uma grande quantidade de dados estruturados geralmente requer recursos de consulta em tempo quase real baseados em índice. Por exemplo, em um cenário de Internet de Veículos (IoV), as informações de manutenção do veículo são consultadas pelo número do veículo. Portanto, as informações do veículo são indexadas com base nos números do veículo quando estão sendo armazenadas, para implementar a resposta de segundo nível nesse cenário. Geralmente, o volume de dados é grande. O usuário pode armazenar dados por um a três anos.

Por exemplo, no setor de IoV, uma empresa de automóveis armazena dados no HBase, que suporta armazenamento no nível PB e consultas CDR em milissegundos.

Figura 4-2 Armazenamento de dados em larga escala na indústria de IoV



O MRS tem as seguintes vantagens neste cenário.

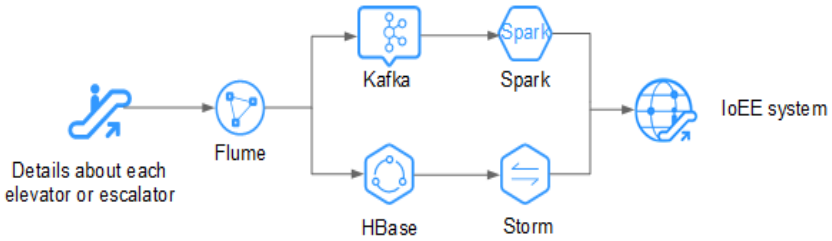
- Em tempo real: Kafka acessa grandes quantidades de mensagens de veículos em tempo real.
- Armazenamento maciço de dados: o HBase armazena grandes volumes de dados e suporta consultas de dados em milissegundos.
- Consulta de dados distribuídos: o Spark analisa e consulta volumes massivos de dados.

Processamento de dados em tempo real

O processamento de dados em tempo real é geralmente usado em cenários como detecção de anomalias, detecção de fraudes, alarmes baseados em regras e monitoramento de processos de serviço. Os dados são processados enquanto estão sendo inseridos no sistema.

Por exemplo, na indústria de elevadores e escadas rolantes (IoEE), os dados de elevadores e escadas rolantes inteligentes são importados para clusters de streaming de MRS em tempo real para alarmes em tempo real.

Figura 4-3 Processamento de streaming de baixa latência na indústria de IoEE



O MRS tem as seguintes vantagens neste cenário.

- Ingestão de dados em tempo real: o Flume implementa a ingestão de dados em tempo real e fornece vários métodos de acesso de coleta e armazenamento de dados.
- Acesso à fonte de dados: Kafka acessa dados de dezenas de milhares de elevadores e escadas rolantes em tempo real.

5 Escolha de uma versão apropriada ao comprar um cluster do MRS

Versões de cluster do MRS

- **Normal**
  - **Funções**

    Esta versão normal fornece operações básicas de cluster, como configuração, gerenciamento e O&M. Para obter detalhes, consulte **Guia de usuário do MapReduce Service**.
  - **Componentes**

    Esta versão normal suporta componentes como Presto, Impala, Kudu, Sqoop e IoTDB. Você pode selecionar componentes de diferentes versões nesta versão. Para obter detalhes, consulte **Lista de versões de componentes do MRS** e **Guia de operação de componente do MapReduce Service**.
- **LTS (suporte de longo prazo)**
  - **Funções**

    Além das operações básicas de cluster, a versão LTS oferece suporte à atualização de versão.

    A versão LTS suporta implantação de multi-AZ.
  - **Componentes**

    A versão LTS suporta o componente HetuEngine além dos componentes suportados pela versão normal. Para obter detalhes, consulte **Guia de operação de componente do MapReduce Service**.

Sugestões de compra

- Clusters da versão LTS suportam atualização de versão. Para tornar os clusters atualizáveis, escolha a versão LTS.
- Clusters da versão LTS podem ser implantados em diferentes AZs para implementar DR entre AZs. Para tornar seus clusters mais seguros e ter recursos de DR mais altos, escolha a versão LTS.
- Clusters da versão LTS suportam o componente HetuEngine. Para usar HetuEngine, escolha a versão LTS.

NOTA

Após a compra, a versão LTS não pode ser alterada para a versão normal. Escolha uma versão apropriada com base nas necessidades da sua empresa.

6 Componentes

6.1 Lista de versões de componentes do MRS

Componentes e versões

Tabela 6-1 lista os componentes e suas versões necessárias para cada versão do cluster do MRS.

NOTA

- O Hadoop inclui HDFS, YARN e MapReduce.
- DBService, ZooKeeper, KrbServer e LdapServer são componentes usados no cluster e não são exibidos durante a criação do cluster.
- As versões de componentes do MRS devem ser consistentes com as versões de componentes de código aberto.

Tabela 6-1 Versões de componentes do MRS

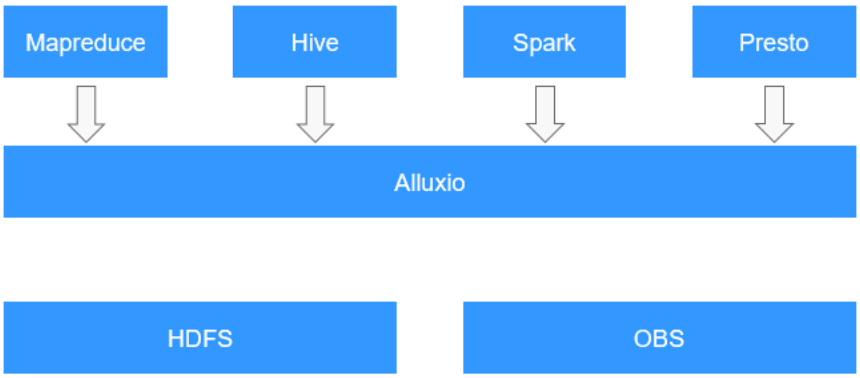
Componente	MRS 1.9.2 (aplicável ao MRS 1.9.*x*)	MRS 3.1.0	MRS 3.1.2-LTS	MRS 3.1.5
Alluxio	2.0.1	-	-	-
CarbonData	1.6.1	2.0.1	2.2.0	2.2.0
ClickHouse	-	21.3.4.25	21.3.4.25	21.3.4.25
DBService	1.0.0	2.7.0	2.7.0	2.7.0
Flink	1.7.0	1.12.0	1.12.2	1.12.2
Flume	1.6.0	1.9.0	1.9.0	1.9.0
HBase	1.3.1	2.2.3	2.2.3	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 (aplicável ao 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 (integrado no 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 é uma tecnologia de orquestração de dados para análises e IA para a nuvem. No ecossistema de Big Data do MRS, o Alluxio fica entre a computação e o armazenamento. Ele fornece uma camada de abstração de dados para estruturas de computação, incluindo MapReduce e Apache Hive, para que aplicações de camada superior possam acessar sistemas de armazenamento persistentes, incluindo HDFS e OBS, por meio de APIs de cliente unificadas e um namespace global. Desta forma, computação e armazenamento são separados.

Figura 6-1 Arquitetura do Alluxio



Vantagens:

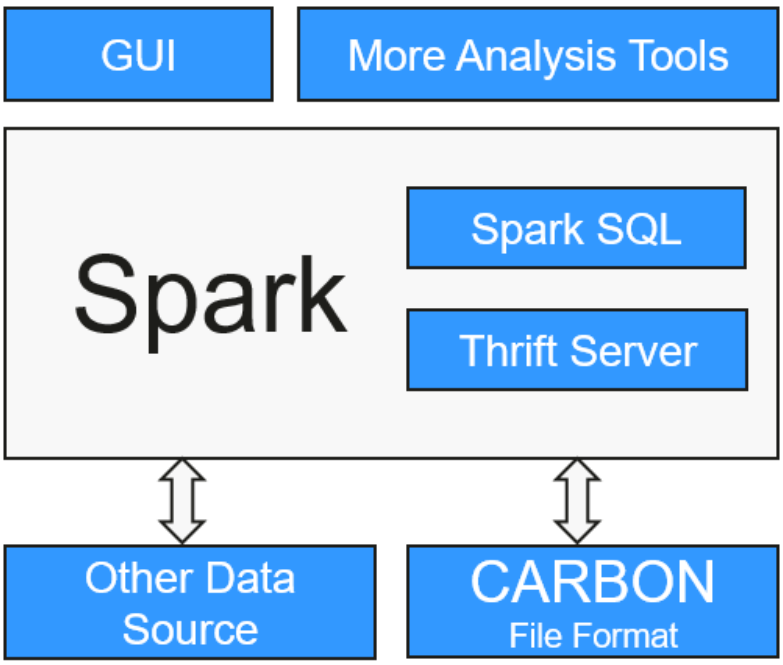
- Fornece taxa de transferência de I/O na memória e torna as aplicações orientadas por dados elásticos econômicos.
- Acesso simplificado ao armazenamento de objetos e à nuvem
- Gerenciamento de dados simplificado e um único ponto de acesso a várias fontes de dados
- Implementação fácil de aplicações

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

6.3 CarbonData

CarbonData é um novo formato de armazenamento de dados nativo do Apache Hadoop. CarbonData permite consultas interativas mais rápidas em PetaBytes de dados usando técnicas avançadas de armazenamento colunar, índice, compactação e codificação para melhorar a eficiência da computação. Além disso, o CarbonData também é um mecanismo de análise de alto desempenho que integra fontes de dados ao Spark.

Figura 6-2 Arquitetura básica do CarbonData



O objetivo do uso do CarbonData é fornecer resposta rápida a consultas ad hoc de Big Data. Essencialmente, o CarbonData é um mecanismo de Processamento Analítico Online (OLAP), que armazena dados usando tabelas semelhantes às do Sistema de Gerenciamento de Banco de Dados Relacional (RDBMS). Você pode importar mais de 10 TB de dados para tabelas criadas no formato CarbonData e o CarbonData organiza e armazena automaticamente os dados usando os índices multidimensionais compactados. Depois que os dados são carregados em CarbonData, o CarbonData responde a consultas ad hoc em segundos.

O CarbonData integra fontes de dados ao ecossistema Spark. Você pode usar o Spark SQL para consultar e analisar dados ou usar a ferramenta de terceiros ThriftServer fornecida pelo Spark para se conectar ao Spark SQL.

Recursos do CarbonData

- SQL: CarbonData é compatível com o Spark SQL e suporta operações de consulta SQL executadas no Spark SQL.
- Definição de conjunto de dados de tabela simples: CarbonData permite definir e criar conjuntos de dados usando instruções de Linguagem de Definição de Dados (DDL). CarbonData DDL é flexível e fácil de usar, e pode definir tabelas complexas.
- Fácil gerenciamento de dados: CarbonData fornece várias funções de gerenciamento de dados para carregamento e manutenção de dados. Ele pode carregar dados históricos e carregar novos dados de forma incremental. Os dados carregados podem ser excluídos de acordo com o tempo de carregamento e as operações específicas de carregamento de dados podem ser canceladas.
- O formato de arquivo CarbonData é um armazenamento colunar no HDFS. Ele tem muitos recursos que um formato colunar moderno tem, como divisível e esquema de compressão.

Recursos únicos do CarbonData

- Armazena dados junto com o índice: acelera significativamente o desempenho da consulta e reduz as varreduras de I/O e os recursos da CPU, quando há filtros na consulta. O índice de CarbonData consiste em vários níveis de índices. Uma estrutura de processamento pode aproveitar esse índice para reduzir a tarefa que precisa agendar e processar, e também pode executar a varredura de pular em uma unidade de grãos mais fina (chamada blocklet) na varredura do lado da tarefa em vez de verificar o arquivo inteiro.
- Dados codificados operáveis: com o suporte a esquemas eficientes de compressão e codificação global, CarbonData pode consultar dados compactados/codificados. Os dados podem ser convertidos pouco antes de retornar os resultados para os usuários, que é "materializado tardiamente".
- Suporta vários casos de uso com um único formato de dados: como consulta interativa no estilo OLAP, Acesso Sequencial (grande varredura) e Acesso Aleatório (varredura estreita).

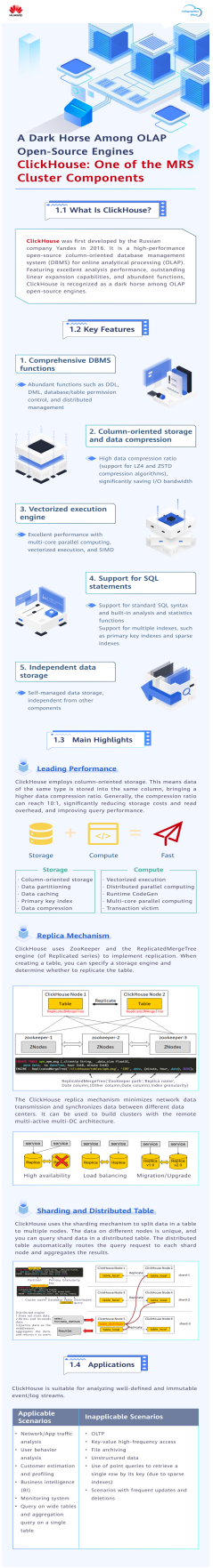
Principais tecnologias e vantagens do CarbonData

- Resposta rápida da consulta: CarbonData apresenta consultas de alto desempenho. A velocidade de consulta do CarbonData é 10 vezes maior que a do Spark SQL. Ele usa formatos de dados dedicados e aplica várias tecnologias de índice, código de dicionário global e várias otimizações push-down, fornecendo resposta rápida a consultas de dados em nível de TB.
- Compressão de dados eficiente: CarbonData comprime dados combinando os algoritmos de compressão leves e pesados. Isso economiza significativamente de 60% a 80% de espaço de armazenamento de dados e o custo de armazenamento de hardware.

Para obter detalhes sobre a arquitetura e os princípios do CarbonData, consulte **https://carbondata.apache.org/**.

6.4 ClickHouse

6.4.1 Infográficos 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

Introdução ao ClickHouse

ClickHouse é um banco de dados colunar de código aberto orientado para análise e processamento on-line. É independente do sistema de Big Data Hadoop e apresenta taxa de compressão final e desempenho de consulta rápida. Além disso, o ClickHouse suporta consulta SQL e oferece bom desempenho, especialmente a análise de agregação e desempenho de consulta com base em tabelas grandes e amplas. A velocidade de consulta é uma ordem de grandeza mais rápida do que a de outros bancos de dados analíticos.

As principais funções do ClickHouse são as seguintes:

Funções abrangentes de DBMS

O ClickHouse é um sistema de gerenciamento de banco de dados (DBMS) que fornece as seguintes funções básicas:

- Linguagem de Definição de Dados (DDL): permite que bancos de dados, tabelas e visualizações sejam criados, modificados ou excluídos dinamicamente sem reiniciar os serviços.
- Linguagem de manipulação de dados (DML): permite que os dados sejam consultados, inseridos, modificados ou excluídos dinamicamente.
- Controle de permissão: suporta configurações de permissão de operação de tabela ou banco de dados baseado no usuário para garantir a segurança dos dados.
- Backup e restauração de dados: suporta backup, exportação, importação e restauração de dados para atender aos requisitos do ambiente de produção.
- Gerenciamento distribuído: fornece o modo de cluster para gerenciar automaticamente vários nós de banco de dados.

Armazenamento baseado em coluna e compressão de dados

O ClickHouse é um banco de dados que usa armazenamento baseado em colunas. Os dados são organizados por coluna. Os dados na mesma coluna são armazenados juntos e os dados em colunas diferentes são armazenados em arquivos diferentes.

Durante a consulta de dados, o armazenamento colunar pode reduzir a faixa de varredura de dados e o tamanho da transmissão de dados, melhorando assim a eficiência da consulta de dados.

Em um sistema tradicional de banco de dados baseado em linhas, os dados são armazenados na sequência em **Tabela 6-2**:

Tabela 6-2 Banco de dados baseado em linha

Linha	ID	Bandeira	Nome	Evento	Tempo
0	12345678901	0	nome1	1	2020/1/11 15:19
1	32345678901	1	nome2	1	2020/5/12 18:10
2	42345678901	1	nome3	1	2020/6/13 17:38

Linha	ID	Bandeira	Nome	Evento	Tempo
N	...	...	...	...	...

Em um banco de dados baseado em linha, os dados na mesma linha são fisicamente armazenados juntos. Em um sistema de banco de dados baseado em colunas, os dados são armazenados na sequência em **Tabela 6-3**:

Tabela 6-3 Base de dados colunar

linha:	0	1	2	N
ID:	12345678901	32345678901	42345678901	...
Bandeira:	0	1	1	...
Nome:	nome1	nome2	nome3	...
Evento:	1	1	1	...
Tempo:	2020/1/11 15:19	2020/5/12 18:10	2020/6/13 17:38	...

Este exemplo mostra apenas a disposição dos dados em um banco de dados colunar. Bancos de dados colunares armazenam dados na mesma coluna juntos e dados em colunas diferentes separadamente. Os bancos de dados colunares são mais adequados para cenários de processamento analítico on-line (OLAP).

Executor vetorizado

O ClickHouse usa os Dados Múltiplos de Instrução Única (SIMD) da CPU para implementar a execução vetorizada. SIMD é um modo de implementação que usa uma única instrução para operar vários pedaços de dados e melhora o desempenho com paralelismo de dados (outros métodos incluem paralelismo no nível de instrução e paralelismo no nível de thread). O princípio dos SIMD é implementar operações de dados paralelas no nível do registrador da CPU.

Modelo relacional e consulta SQL

O ClickHouse usa SQL como linguagem de consulta e fornece APIs de consulta SQL padrão para integração fácil com o ClickHouse.

Além disso, o ClickHouse utiliza um modelo relacional. Portanto, o custo de migrar o sistema construído em um banco de dados relacional e tradicional ou armazém de dados para ClickHouse é menor.

Fragmentação de dados e consulta distribuída

O cluster de ClickHouse consiste em um ou mais partições, e cada partição corresponde a um nó de serviço ClickHouse. O número máximo de partições depende do número de nós (uma partição corresponde a apenas um nó de serviço).

ClickHouse apresenta os conceitos de tabela local e tabela distribuída. Uma tabela local é equivalente a uma partição de dados. Uma tabela distribuída em si não armazena nenhum

dado. É um proxy de acesso da tabela local e funciona como o middleware de fragmentação. Com a ajuda de tabelas distribuídas, vários fragmentos de dados podem ser acessados usando o proxy, implementando assim a consulta distribuída.

Aplicações ClickHouse

ClickHouse é a abreviação de Click Stream e Data Warehouse. Ele é inicialmente aplicado a uma ferramenta de análise de tráfego da Web para realizar análise OLAP para armazéns de dados com base em fluxos de eventos de clique em página. Atualmente, o ClickHouse é amplamente utilizado em campos de publicidade na Internet, análise de tráfego de aplicações e Web, telecomunicações, finanças e Internet das Coisas (IoT). É aplicável a cenários de aplicações de business intelligence e possui um grande número de aplicações e práticas em todo o mundo. Para mais detalhes, visite **https://clickhouse.tech/docs/en/introduction/adopters/**.

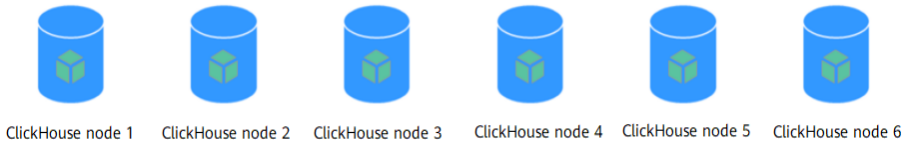
Recursos de código aberto aprimorados de ClickHouse

O ClickHouse MRS tem vantagens como modo de cluster automático, implantação de HA e dimensionamento suave e elástico.

- Modo de cluster automático

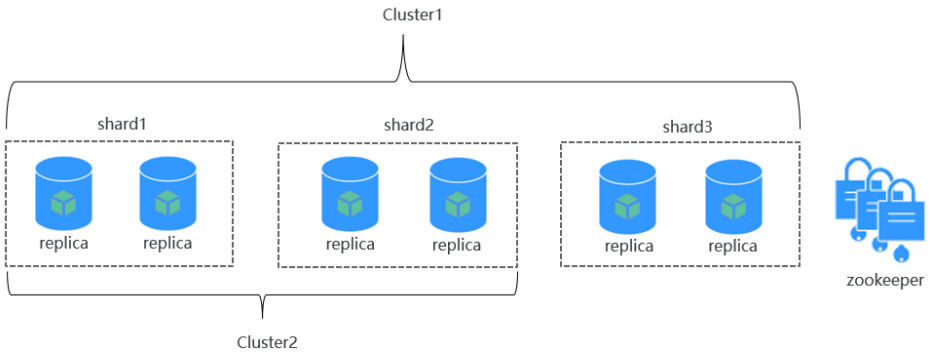
  Como mostrado em **Figura 6-3**, um cluster consiste em vários nós de ClickHouse, que não tem nenhum nó central. É mais um pool de recursos estáticos. Se o modo de cluster de ClickHouse for usado para serviços, será necessário pré-definir as informações do cluster no arquivo de configuração de cada nó. Somente assim, os serviços podem ser acessados corretamente.

  Figura 6-3 Cluster de ClickHouse

  Os usuários desconhecem as partições de dados e o armazenamento de réplicas em sistemas de banco de dados comuns. No entanto, o ClickHouse permite planejar e definir proativamente configurações detalhadas, como locais de fragmentos, partições e réplica. A instância de ClickHouse do MRS embala o trabalho de forma unificada e adapta-o ao modo automático, implementando um gerenciamento unificado, flexível e fácil de usar. Uma instância do ClickHouse consiste em três nós do ZooKeeper e vários nós do ClickHouse. O modo de réplica dedicada é usado para garantir alta confiabilidade de cópias de dados duplas.

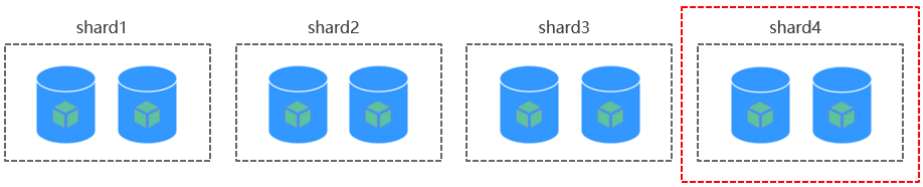
Figura 6-4 Estrutura do cluster do ClickHouse



- Dimensionamento suave e elástico

  À medida que os negócios crescem rapidamente, o MRS fornece ClickHouse para cenários como capacidade de armazenamento do cluster ou recursos de computação da CPU que se aproximam do limite. Esta ferramenta é usada para migrar algumas partições de uma ou várias tabelas de MergeTree em vários nós de ClickHouseServer para as mesmas tabelas em outros nós de ClickHouseServer. Dessa forma, a disponibilidade do serviço é garantida e a expansão suave da capacidade é implementada.

  Ao adicionar nós de ClickHouse a um cluster, use essa ferramenta para migrar alguns dados dos nós existentes para os novos para balanceamento de dados após a expansão.

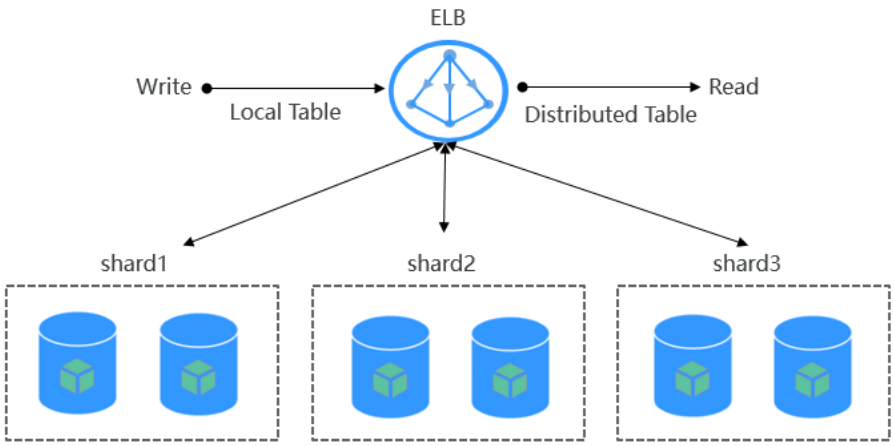


- Arquitetura de implementação de HA

  O MRS usa a arquitetura de implementação de alta disponibilidade (HA) baseada em ELB para distribuir automaticamente o tráfego de acesso do usuário para vários nós de back-end, expandindo os recursos de serviço para sistemas externos e melhorando a tolerância a falhas. Conforme mostrado em **Figura 6-5**, quando uma aplicação de cliente solicita um cluster, o Elastic Load Balance (ELB) é usado para distribuir o tráfego. Com o mecanismo de sondagem do ELB, os dados são gravados em tabelas locais e lidos de tabelas distribuídas em diferentes nós. Desta forma, a carga de leitura/gravação de dados e a alta disponibilidade de acesso à aplicação são garantidas.

  Depois que o cluster de ClickHouse é provisionado, cada nó de instância de ClickHouse no cluster corresponde a uma réplica e duas réplicas formam uma partição lógica. Por exemplo, ao criar uma tabela de ReplicatedMergeTree, você pode especificar partições para que os dados possam ser sincronizados automaticamente entre duas réplicas na mesma partição.

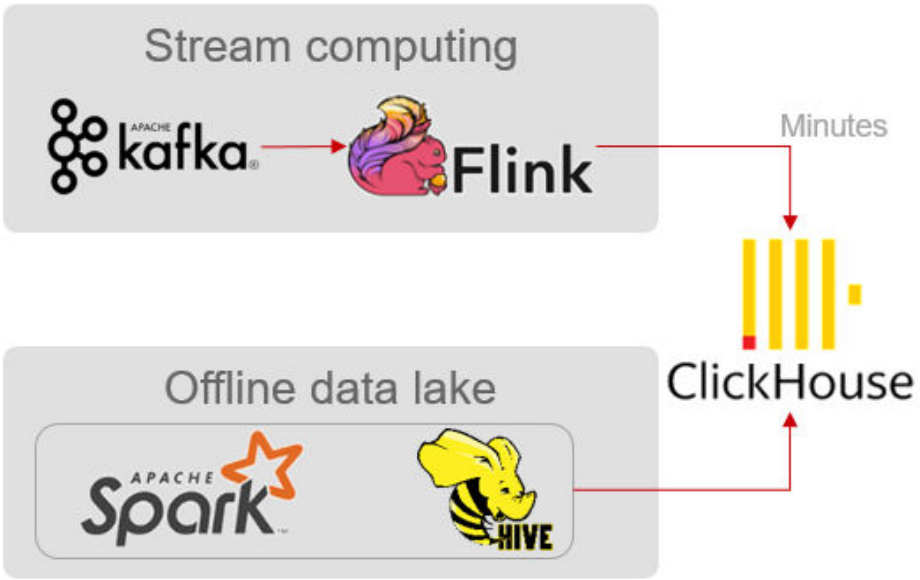
Figura 6-5 Arquitetura de implementação de HA



Relação entre ClickHouse e outros componentes

O ClickHouse depende do ZooKeeper para instalação e implementação.

As aplicações de computação de fluxo do Flink são usadas para gerar dados de relatórios comuns (tabelas detalhadas) e gravar os dados do relatório em ClickHouse em tempo quase real. Os jobs de Hive/Spark são usados para gerar dados de relatório comuns (tabelas detalhadas) e importar dados para ClickHouse em lote.



NOTA

Atualmente, o ClickHouse não oferece suporte à interconexão com o Kafka no modo normal ou com o HDFS no modo de segurança.

6.5 DBService

6.5.1 Princípios básicos do DBService

Visão geral

DBService é um sistema de armazenamento HA para bancos de dados relacionais, que é aplicável ao cenário em que uma pequena quantidade de dados (cerca de 10 GB) precisa ser armazenada, por exemplo, metadados de componentes. O DBService só pode ser usado por componentes internos de um cluster e fornece funções de armazenamento de dados, consulta e exclusão.

DBService é um componente básico de um cluster. Componentes como Hive, Hue, Oozie, Loader e Redis, e Loader armazenam seus metadados no DBService e fornecem as funções de backup e restauração de metadados usando DBService.

Arquitetura de DBService

O DBService no cluster funciona no modo ativo/em espera. Duas instâncias do DBServer são implementadas e cada instância contém três módulos: HA, Database e FloatIP.

Figura 6-6 mostra a arquitetura lógica do DBService.

Figura 6-6 Arquitetura de DBService

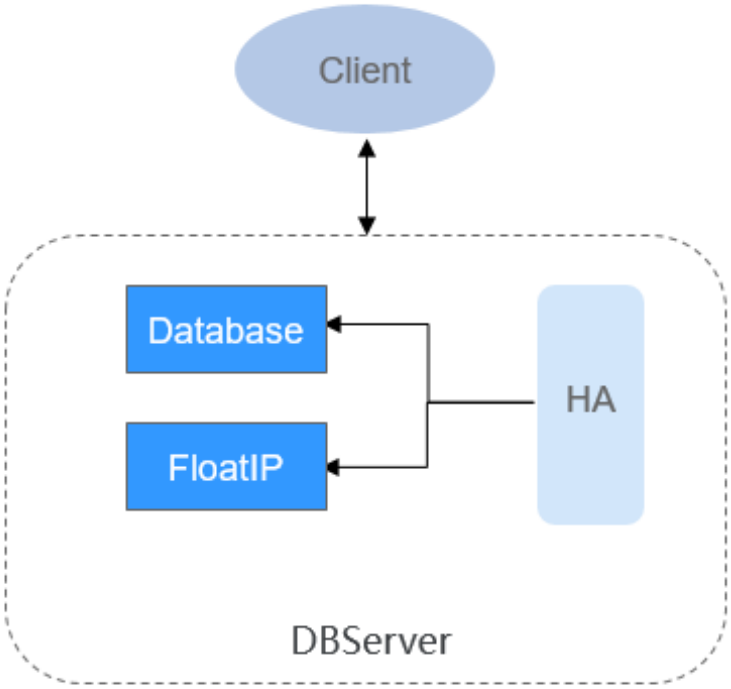


Tabela 6-4 descreve os módulos mostrados em **Figura 6-6**.

Tabela 6-4 Descrição do módulo

Nome	Descrição
HA	Módulo de gestão de HA. O DBServer ativo/em espera usa o módulo HA para gerenciamento.
Database	Módulo de base de dados. Este módulo armazena os metadados do módulo Client.
FloatIP	Endereço IP flutuante que fornece a função de acesso externamente. Ele é ativado somente na instância de DBServer ativa e é usado pelo módulo Client para acessar o Database.
Client	Cliente que usa o componente de DBService, que é implementado no nó da instância do componente. O cliente se conecta ao banco de dados usando FloatIP e, em seguida, executa operações de adição, exclusão e modificação de metadados.

6.5.2 Relação entre DBService e outros componentes

DBService é um componente básico de um cluster. Componentes como Hive, Hue, Oozie, Loader, Metadados e Redis, e Loader armazenam seus metadados no DBService e fornecem as funções de backup e restauração de metadados usando DBService.

6.6 Flink

6.6.1 Princípios básicos do Flink

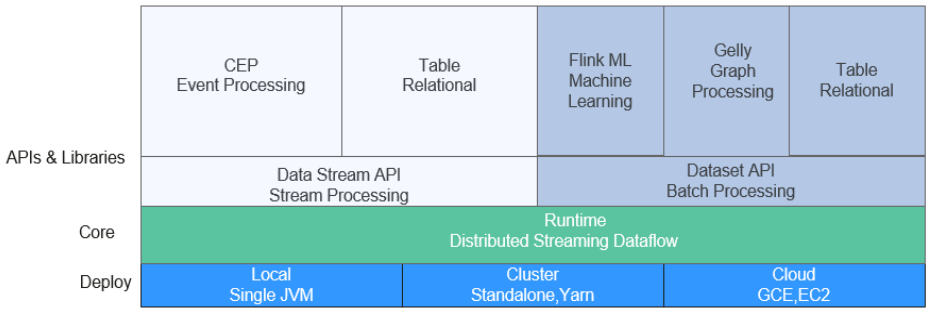
Visão geral

Flink é uma estrutura de computação unificada que suporta processamento em lote e processamento de fluxo. Ele fornece um mecanismo de processamento de dados de fluxo que suporta distribuição de dados e computação paralela. Flink apresenta processamento de fluxo e é um mecanismo de processamento de fluxo de código aberto superior na indústria.

O Flink oferece processamento de dados de pipeline de alta simultaneidade, latência de milissegundos e alta confiabilidade, tornando-o extremamente adequado para processamento de dados de baixa latência.

Figura 6-7 mostra a pilha de tecnologia do Flink.

Figura 6-7 Pilha de tecnologia de Flink



O Flink fornece os seguintes recursos na versão atual:

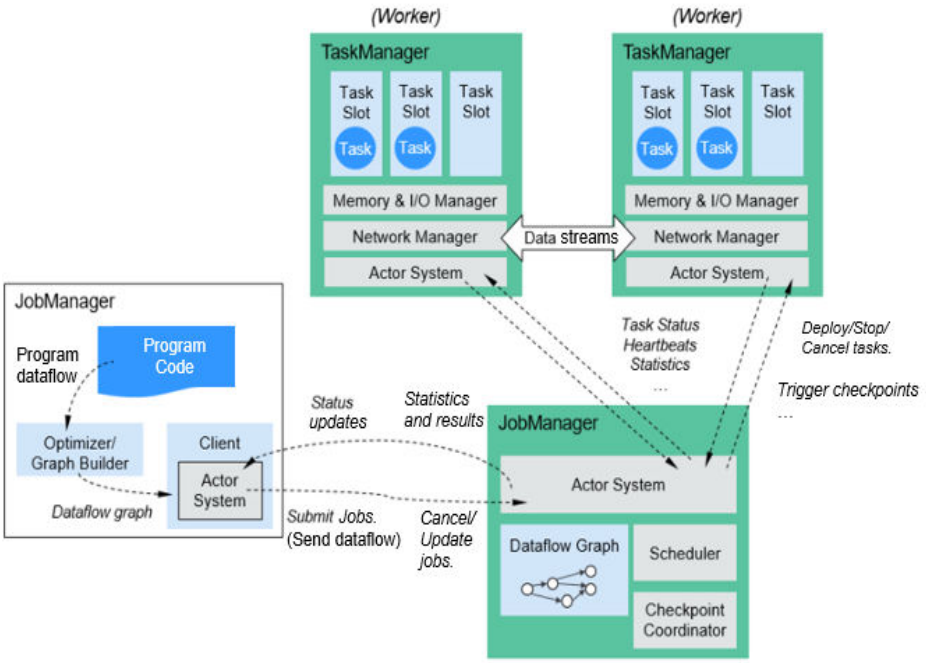
- DataStream
- Ponto de verificação
- Janela
- Pipeline de job
- Tabela de configuração

Outros recursos são herdados da comunidade de código aberto e não são aprimorados. Para obter detalhes, visite **https://ci.apache.org/projects/flink/flink-docs-release-1.12/**.

Arquitetura do Flink

Figura 6-8 mostra a arquitetura do Flink.

Figura 6-8 Arquitetura do Flink



Como mostrado na figura acima, todo o sistema do Flink consiste em três partes:

- Client

  O cliente do Flink é usado para enviar jobs (jobs de streaming) para o Flink.

- TaskManager

  TaskManager é um nó de execução de serviço do Flink. Ele executa tarefas específicas. Um sistema do Flink pode ter múltiplos TaskManagers. Estes TaskManagers são equivalentes entre si.

- JobManager

  JobManager é um nó de gerenciamento do Flink. Ele gerencia todos os TaskManagers e agenda tarefas enviadas pelos usuários para TaskManagers específicos. No modo de alta disponibilidade (HA), vários JobManagers são implementados. Entre esses JobManagers um é selecionado como o JobManager ativo e os outros são em espera.

Para obter mais informações sobre a arquitetura do Flink, visite **https://ci.apache.org/projects/flink/flink-docs-master/docs/concepts/flink-architecture/**.

Princípios do Flink

- **Fluxo & Transformação & Operador**

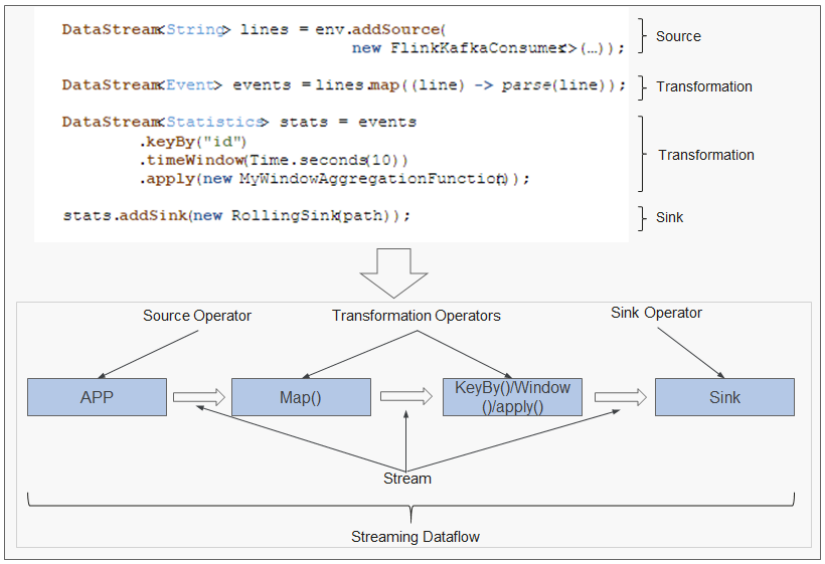
  Um programa Flink consiste em dois blocos de construção: fluxo e transformação.

  a. Conceitualmente, um fluxo é um fluxo (potencialmente interminável) de registros de dados, e uma transformação é uma operação que leva um ou mais fluxos como entrada e produz um ou mais fluxos de saída como resultado.

  b. Quando um programa Flink é executado, ele é mapeado para um fluxo de dados de streaming. Um fluxo de dados de streaming consiste em um grupo de fluxos e operadores de transformação. Cada fluxo de dados começa com um ou mais operadores de origem e termina em um ou mais operadores de coletor. Um fluxo de dados se assemelha a um grafo acíclico direcionado (DAG).

  Figura 6-9 mostra o fluxo de dados de fluxo contínuo para o qual um programa Flink é mapeado.

  Figura 6-9 Exemplo de DataStream do Flink



  Como mostrado em **Figura 6-9**, **FlinkKafkaConsumer** é um operador de origem; Map, KeyBy, TimeWindow e Apply são operadores de transformação; RollingSink é um operador de coletor.

- **Fluxo de dados do pipeline**

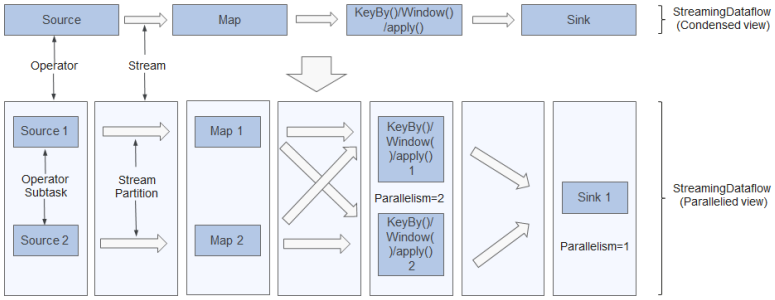
  As aplicações no Flink podem ser executadas em modo paralelo ou distribuído. Um fluxo pode ser dividido em uma ou mais partições de fluxo, e um operador pode ser dividido em várias subtarefas de operador.

  O executor de fluxos e operadores são otimizados automaticamente com base na densidade de operadores a montante e a jusante.

  - Operadores com baixa densidade não podem ser otimizados. Cada subtarefa de operador é executada separadamente em threads diferentes. O número de subtarefas de operador é o paralelismo desse operador em particular. O paralelismo (o número total de partições) de um fluxo é o de seu operador produtor. Diferentes operadores

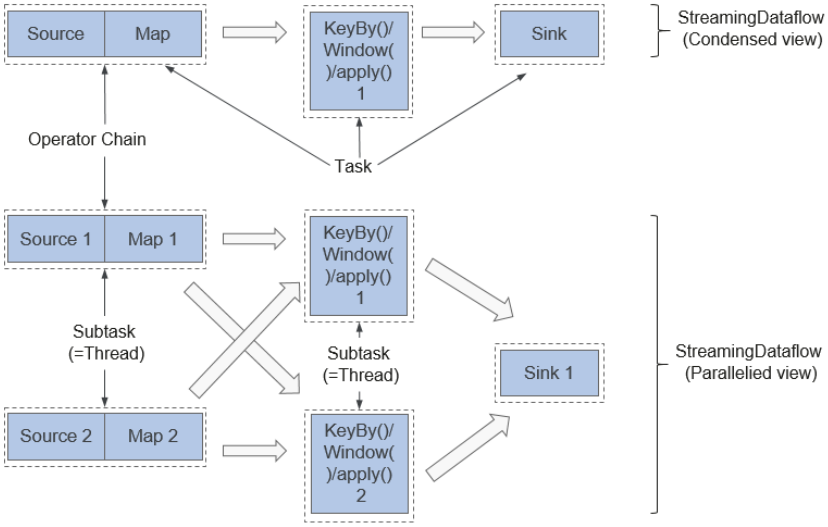
do mesmo programa podem ter diferentes níveis de paralelismo, como mostrado em **Figura 6-10**.

Figura 6-10 Operador



- Operadores com alta densidade podem ser otimizados. Flink encadeia subtarefas do operador em uma tarefa, ou seja, uma cadeia de operadores. Cada cadeia de operadores é executada por um thread no TaskManager, como mostrado em **Figura 6-11**.

Figura 6-11 Cadeia do operador



  - Na parte superior do **Figura 6-11**, os operadores condensados Source e Map são encadeados em uma Operator Chain, ou seja, um operador maior. A Operator Chain, o KeyBy e o Sink representam um operador, respectivamente, e estão conectados entre si através de fluxos. Cada operador corresponde a uma tarefa durante a execução. Ou seja, existem três tarefas na parte superior.
  - Na parte inferior de **Figura 6-11**, cada tarefa, exceto Sink, é paralela em duas subtarefas. O paralelismo do operador Sink é um.

Principais recursos

- Processamento de fluxo

  O mecanismo de processamento de fluxo em tempo real apresenta alta taxa de transferência, alto desempenho e baixa latência, o que pode fornecer capacidade de processamento em milissegundos.

- Vários gerenciamento de status

  A aplicação de processamento de fluxo precisa armazenar os eventos recebidos ou o resultado intermediário em um determinado período de tempo para acesso e processamento subsequentes em um determinado ponto de tempo. O Flink fornece diversos recursos para gerenciamento de status, incluindo:

  - Vários tipos básicos de status: o Flink fornece vários estados para estruturas de dados, como ValueState, ListState e MapState. Os usuários podem selecionar o tipo de status mais eficiente e adequado com base no modelo de serviço.
  - Back-end de estado rico: o Back-end de estado gerencia o status das aplicações e executa as operações de Checkpoint conforme necessário. O Flink fornece back-ends de estados diferentes. O estado pode ser armazenado na memória ou no RocksDB e suporta o mecanismo Ponto de verificação assíncrono e incremental.
  - Consistência de estado exatamente uma vez: os recursos de Ponto de verificação e recuperação de falhas do Flink garantem que o status das tarefas da aplicação seja consistente antes e depois da ocorrência de uma falha. O Flink suporta saída transacional para alguns dispositivos de armazenamento específicos. Desta forma, a saída de exatamente uma vez pode ser garantida mesmo quando ocorre uma falha.

- Diversas semânticas de tempo

  O tempo é uma parte importante das aplicações de processamento de fluxo. Para aplicações de processamento de fluxo em tempo real, operações como agregação de janelas, detecção e correspondência com base na semântica de tempo são muito comuns. Flink fornece várias semânticas de tempo.

  - Horário do evento: o carimbo de data/hora fornecido pelo evento é usado para o cálculo, facilitando o processamento dos eventos que chegam em uma sequência aleatória ou chegam atrasados.
  - Marca d'água: o Flink introduz o conceito de Marca d'água para medir o desenvolvimento do tempo do evento. A marca d'água também fornece garantia flexível para balancear a latência do processamento e a integridade dos dados. Ao processar fluxos de eventos com marca d'água, o Flink oferece várias opções de processamento se os dados chegarem após o cálculo, por exemplo, redirecionando dados (saída lateral) ou atualizando o resultado do cálculo.
  - O tempo de processamento e o tempo de ingestão são suportados.
  - Janela de streaming altamente flexível: o Flink suporta a janela de tempo, a janela de contagem, a janela de sessão e a janela personalizada orientada por dados. Você pode personalizar as condições de disparo para implementar o modo de cálculo de streaming complexo.

- Mecanismo de tolerância a falhas

  Em um sistema distribuído, se uma única tarefa ou nó quebrar ou estiver com defeito, toda a tarefa pode falhar. O Flink fornece um mecanismo de tolerância a falhas no nível da tarefa, que garante que os dados do usuário não sejam perdidos quando uma exceção ocorrer em uma tarefa e possam ser restaurados automaticamente.

  - Ponto de verificação: o Flink implementa a tolerância a falhas com base no ponto de verificação. Os usuários podem personalizar a política de pontos de verificação para toda a tarefa. Quando uma tarefa falha, ela pode ser restaurada para o status do ponto de verificação mais recente e para os dados após o snapshot ser reenviado da fonte de dados.
  - Ponto de salvamento: um ponto de salvamento é um instantâneo consistente do status da aplicação. O mecanismo de ponto de salvamento é semelhante ao do ponto de verificação. No entanto, o mecanismo de ponto de salvamento precisa ser

acionado manualmente. O mecanismo de ponto de salvamento garante que as informações de status da aplicação de fluxo atual não sejam perdidas durante a atualização ou migração da tarefa, facilitando a suspensão e a recuperação da tarefa em qualquer ponto de tempo.

- Flink SQL

  APIs de tabela e SQL usam o Apache Calcite para analisar, verificar e otimizar consultas. APIs de tabela e SQL podem ser perfeitamente integradas com APIs de DataStream e DataSet e suportam funções escalares definidas pelo usuário, funções de agregação e funções de valor de tabela. A definição de aplicações como análise de dados e ETL é simplificada. O exemplo de código a seguir mostra como usar instruções Flink SQL para definir uma aplicação de contagem que registra horários de sessão.

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

  Para obter mais informações sobre o Flink SQL, consulte **https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html**.

- CEP em SQL

  O Flink permite que os usuários representem resultados de consultas de processamento de eventos complexos (CEP) em SQL para correspondência de padrões e avaliem fluxos de eventos no Flink.

  O CEP SQL é implementado através da sintaxe SQL **MATCH_RECOGNIZE**. A cláusula **MATCH_RECOGNIZE** é suportada pelo Oracle SQL desde o Oracle Database 12c e é usada para indicar a correspondência de padrões de eventos em SQL. Veja a seguir um exemplo 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 Solução HA do Flink

Solução HA do Flink

Um cluster de Flink tem apenas um JobManager. Isso tem os riscos de pontos únicos de falhas (SPOFs). Existem três modos de Flink: Flink On Yarn, Flink Standalone e Flink Local. Os modos Flink On Yarn e Flink Standalone são baseados em clusters e o modo Flink Local é baseado em um único nó. Flink On Yarn e Flink Standalone fornecem um mecanismo de HA. Com esse mecanismo, você pode recuperar o JobManager de falhas e, assim, eliminar os riscos de SPOF. Esta seção descreve o mecanismo de HA do Flink On Yarn.

O Flink suporta o modo de HA e a recuperação de exceção de job que dependem muito do ZooKeeper. Se você quiser habilitar as duas funções, configure ZooKeeper no arquivo **flink-conf.yaml** com antecedência da seguinte forma:


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

Flink On Yarn

Flink JobManager e Yarn ApplicationMaster estão no mesmo processo. Yarn ApplicationMaster monitora ApplicationMaster. Se o ApplicationMaster estiver anormal, o Yarn o reiniciará e restaurará todos os metadados do JobManager do HDFS. Durante a recuperação, as tarefas existentes não podem ser executadas e as novas tarefas não podem ser enviadas. O ZooKeeper armazena metadados do JobManager, como informações sobre jobs, para serem usados pelo novo JobManager. Uma falha do TaskManager é ouvida e processada pelo mecanismo DeathWatch do Akka no JobManager. Quando um TaskManager falha, um contêiner é solicitado novamente do Yarn e um TaskManager é criado.

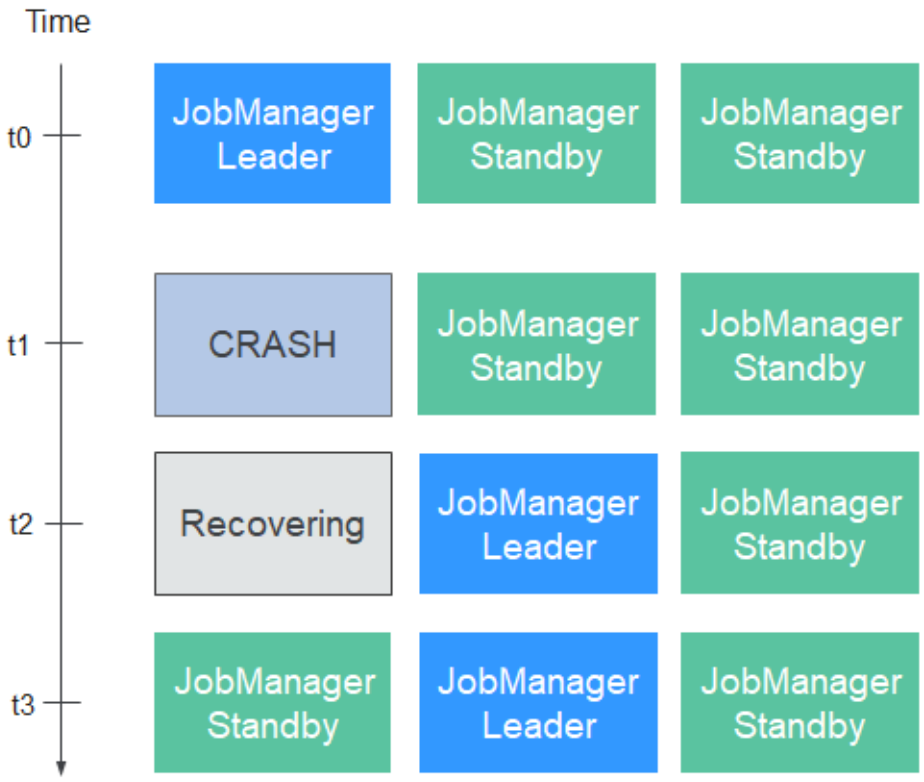
Para obter mais informações sobre a solução de HA do Flink no YARN, visite:

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

Standalone

No modo standalone, vários JobManagers podem ser iniciados e o ZooKeeper elege um como o líder JobManager. Neste modo, há um líder JobManager e vários JobManagers em espera. Se o JobManager líder falhar, um JobManager de espera assume a liderança. **Figura 6-12** mostra o processo de alternância de um JobManager líder/em espera.

Figura 6-12 Processo de alternância



Restauração de TaskManager

Uma falha do TaskManager é ouvida e processada pelo mecanismo DeathWatch do Akka no JobManager. Se o TaskManager falhar, o JobManager criará um TaskManager e migrará os serviços para o TaskManager criado.

Restauração de JobManager

Flink JobManager e Yarn ApplicationMaster estão no mesmo processo. Yarn ApplicationMaster monitora ApplicationMaster. Se o ApplicationMaster estiver anormal, o Yarn o reiniciará e restaurará todos os metadados do JobManager do HDFS. Durante a recuperação, as tarefas existentes não podem ser executadas e as novas tarefas não podem ser enviadas.

Restauração de jobs

Se você quiser restaurar jobs, certifique-se de que a política de inicialização esteja configurada nos arquivos de configuração do Flink. As políticas de reinicialização suportadas são **fixed-delay**, **failure-rate** e **none**. Jobs só podem ser restauradas quando a política estiver configurada para **fixed-delay** ou **failure-rate**. Se a política de reinicialização estiver configurada para **none** e o ponto de verificação estiver configurado para jobs, a política de reinicialização será configurada automaticamente para **fixed-delay** e o valor de **restart-strategy.fixed-delay.attempts** (que especifica o número de vezes de repetição) será configurado para **Integer.MAX_VALUE**.

Para obter detalhes sobre as três estratégias, visite o site oficial do Flink em **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/task_failure_recovery.html**. As estratégias de configuração são as seguintes:

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

Os jobs serão restaurados nos seguintes cenários:

- Se um JobManager falhar, todos os seus jobs serão interrompidos e recuperados após outro JobManager ser criado e executado.
- Se um TaskManager falhar, todas as tarefas no TaskManager serão interrompidas e iniciadas até que haja recursos disponíveis.
- Quando um job de um job falha, o job é reiniciado.

  NOTA

  Para obter detalhes sobre como configurar estratégias de reinicialização de jobs, consulte **https://ci.apache.org/projects/flink/flink-docs-release-1.12/ops/jobmanager_high_availability.html**.

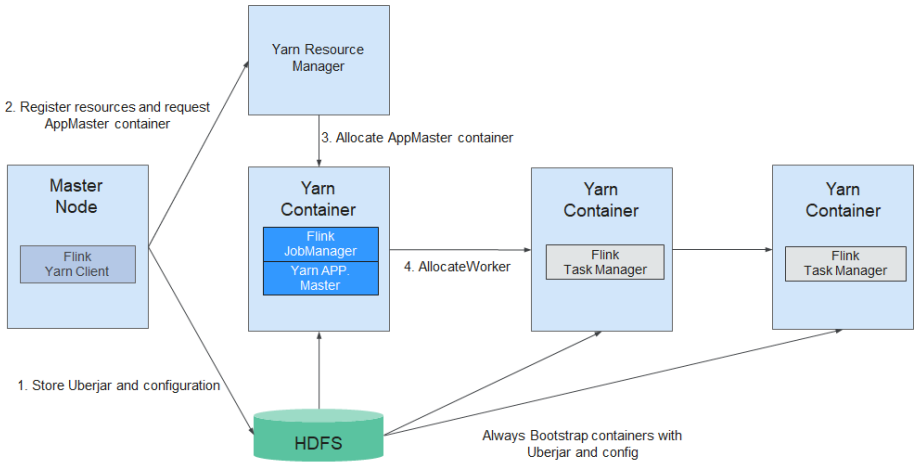
6.6.3 Relação entre Flink e outros componentes

Relação entre Flink e YARN

O Flink suporta o modo de gerenciamento de cluster baseado em YARN. Neste modo, o Flink serve como uma aplicação do YARN e é executado no YARN.

Figura 6-13 mostra a implementação do cluster Flink baseado em YARN.

Figura 6-13 Implementação de cluster de Flink baseado em YARN



1. O cliente de Flink YARN primeiro verifica se há recursos suficientes para iniciar o cluster de YARN. Se sim, o cliente de Flink YARN carrega arquivos JAR e arquivos de configuração para o HDFS.
2. O cliente de Flink YARN se comunica com o ResourceManager do YARN para solicitar um contêiner para iniciar ApplicationMaster. Depois que todos os NodeManagers do YARN terminarem de baixar o arquivo JAR e os arquivos de configuração, o ApplicationMaster será iniciado.
3. Durante a inicialização, o ApplicationMaster interage com o ResourceManager do YARN para solicitar o contêiner para iniciar um TaskManager. Depois que o contêiner estiver pronto, o processo de TaskManager é iniciado.
4. No cluster de Flink YARN, os ApplicationMaster e JobManager do Flink estão sendo executados no mesmo contêiner. O ApplicationMaster informa a cada TaskManager o endereço RPC do JobManager. Depois que os TaskManagers são iniciados, eles se registram no JobManager.
5. Depois que todos os TaskManagers for registrado no JobManager, o Flink será inicializado no cluster do YARN. O cliente de Flink YARN pode enviar jobs do Flink ao JobManager e o Flink pode mapear, agendar e calcular os jobs.

6.6.4 Recursos de código aberto aprimorados do Flink

6.6.4.1 Janela

Recurso de código aberto aprimorado: janela

Esta seção descreve a janela deslizante do Flink e fornece o método de otimização da janela deslizante. Para obter detalhes sobre janelas, visite o site oficial em **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/stream/operators/windows.html**.

Introdução à janela

Os dados em uma janela são salvos como resultados intermediários ou dados originais. Se você executar uma operação de soma (**window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).sum**) sobre os dados na janela, apenas o resultado intermediário será retido. Se uma janela personalizada

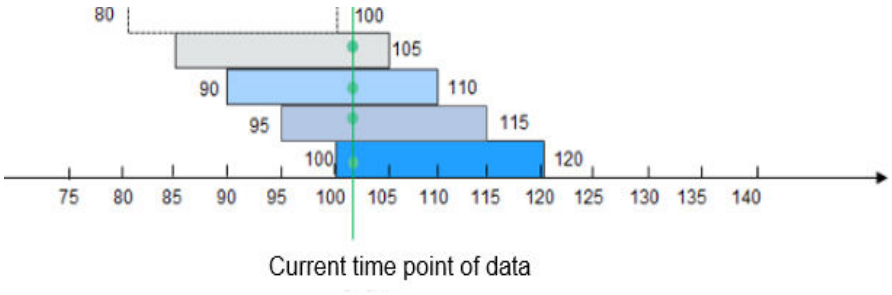
(**window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).apply(new UDF)**) for usada, todos os dados originais na janela serão salvos.

Se as janelas personalizadas **SlidingEventTimeWindow** e **SlidingProcessingTimeWindow** forem usadas, os dados serão salvos como vários backups. Suponha que a janela é definida da seguinte forma:

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

Se um bloco de dados chega, ele é atribuído a quatro janelas diferentes (20/5 = 4). Ou seja, os dados são salvos como quatro cópias na memória. Quando o tamanho da janela ou o período de deslizamento é definido como um valor grande, os dados serão salvos como cópias excessivas, causando redundância.

Figura 6-14 Estrutura original de uma janela



Se um bloco de dados chega ao 102º segundo, ele é atribuído às janelas [85, 105), [90, 110), [95, 115) e [100, 120).

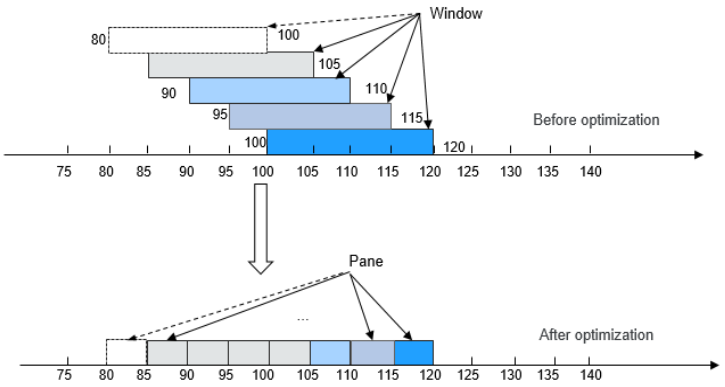
Otimização de janela

Como mencionado no anterior, há cópias de dados excessivas quando os dados originais são salvos em SlidingEventTimeWindow e SlidingProcessingTimeWindow. Para resolver esse problema, a janela que armazena os dados originais é reestruturada, o que otimiza o armazenamento e reduz muito o espaço de armazenamento. O esquema de otimização de janela é o seguinte:

1. Use o período deslizante como uma unidade para dividir uma janela em painéis diferentes.

   Uma janela consiste em um ou vários painéis. Um painel é essencialmente um período deslizante. Por exemplo, o período de deslizamento (ou seja, o painel) de **window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds.of(5)))** dura 5 segundos. Se essa janela variar de [100, 120), ela pode ser dividida em painéis [100, 105), [105, 110), [110, 115) e [115, 120).

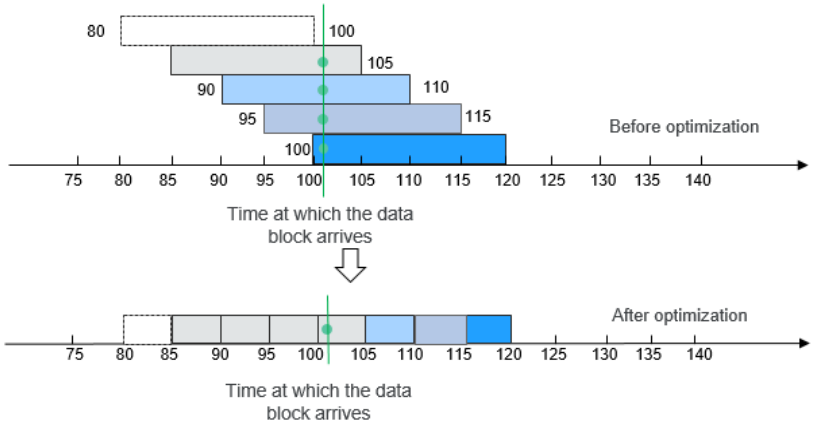
Figura 6-15 Otimização de janela



2. Quando um bloco de dados chega, ele não é atribuído a uma janela específica. Em vez disso, o Flink determina o painel ao qual o bloco de dados pertence com base no carimbo de data/hora do bloco de dados e salva o bloco de dados no painel.

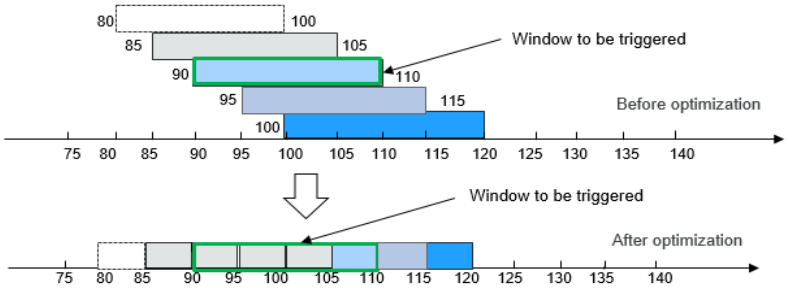
   Um bloco de dados é salvo apenas em um painel. Neste caso, apenas uma cópia de dados existe na memória.

Figura 6-16 Salvar dados em uma janela



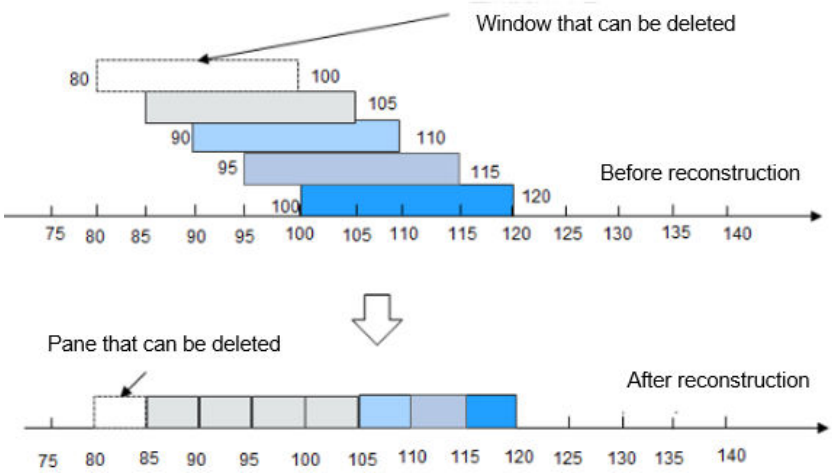
3. Para acionar uma janela, calcule todos os painéis contidos na janela e combine todos esses painéis em uma janela completa.

Figura 6-17 Acionar uma janela



4. Se um painel não for necessário, você poderá excluí-lo da memória.

Figura 6-18 Excluir uma janela



Após a otimização, a quantidade de cópias de dados na memória e no snapshot é bastante reduzida.

6.6.4.2 Pipeline de job

Recurso de código aberto aprimorado: pipeline de job

Geralmente, o código lógico relacionado a um serviço é armazenado em um grande pacote JAR, que é chamado de Fat JAR. As desvantagens do Fat JAR são as seguintes:

- Quando a lógica de serviço se torna cada vez mais complexa, o tamanho do Fat JAR aumenta.
- Fat Jar torna a coordenação complexa. Os desenvolvedores de todos os serviços estão trabalhando com a mesma lógica de serviço. Mesmo que a lógica de serviço possa ser dividida em vários módulos, todos os módulos são fortemente acoplados uns com os outros. Se o requisito precisa ser alterado, todo o diagrama de fluxo precisa ser replanejado.

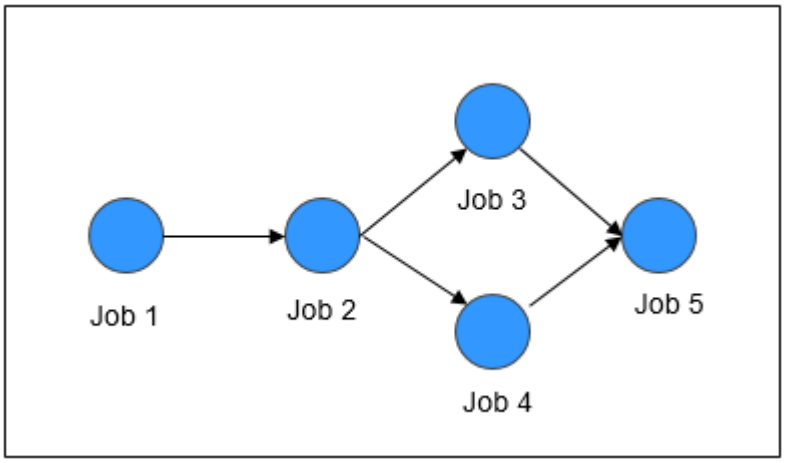
A divisão de jobs enfrenta os seguintes problemas:

- A transmissão de dados entre jobs pode ser alcançada usando Kafka. Por exemplo, o job A transmite dados para o tópico A no Kafka e, em seguida, o job B e o job C lêem dados do tópico A no Kafka. Esta solução é simples e fácil de implementar, mas a latência é sempre superior a 100 ms.
- Os operadores são conectados usando o protocolo TCP. No ambiente distribuído, os operadores podem ser agendados para qualquer nó e os serviços upstream e downstream não podem detectar o agendamento.

Pipeline de trabalho

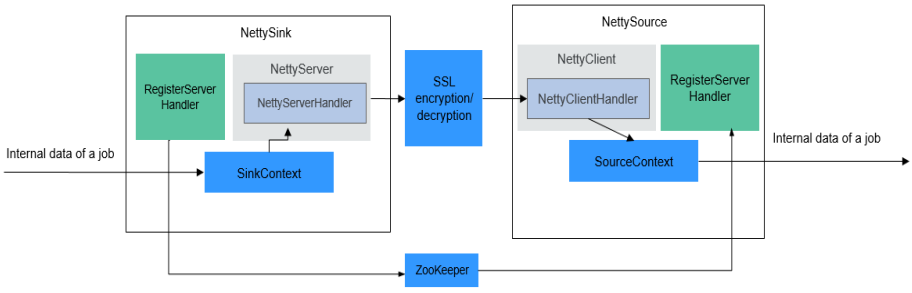
Um pipeline consiste em vários jobs do Flink conectados por meio de TCP. Os jobs de upstream podem enviar dados para jobs de downstream. O diagrama de fluxo sobre a transmissão de dados é chamado de pipeline de job, como mostrado em **Figura 6-19**.

Figura 6-19 Pipeline de job



Princípios do pipeline de job

Figura 6-20 Princípios do pipeline de job



- NettySink e NettySource

  Em um pipeline, os jobs upstream e downstream se comunicam uns com os outros por meio de Netty. O operador Sink do job upstream funciona como um servidor e o operador Source do job downstream funciona como um cliente. O operador Sink do job upstream é chamado NettySink e o operador Source do job downstream é chamado NettySource.

- NettyServer e NettyClient

  O NettySink funciona como o servidor de Netty. No NettySink, o NettyServer cumpre a função de um servidor. NettySource é o cliente de Netty. Em NettySource, o NettyClient realiza a função de um cliente.

- Publicador

  O job que envia dados para jobs downstream por meio do NettySink é chamado de publicador.

- Assinante

  O job que recebe dados de jobs upstream por meio do NettySource é chamado de assinante.

- RegisterServer

  RegisterServer é a memória de terceiros que armazena o endereço IP, o número da porta e as informações de simultaneidade sobre NettyServer.

- A arquitetura geral de fora para dentro é a seguinte:
  - NettySink->NettyServer->NettyServerHandler
  - NettySource->NettyClient->NettyClientHandler

Funções do pipeline de job

- **NettySink**

  O NettySink é composto pelos seguintes módulos principais:

  - RichParallelSinkFunction

    NettySink herda RichParallelSinkFunction e atributos dos operadores Sink. A API de RichParallelSinkFunction implementa as seguintes funções:

    - Inicia o operador NettySink.
    - Executa o operador NettySink e recebe dados do operador upstream.
    - Cancela a execução de operadores NettySink.

    As seguintes informações podem ser obtidas usando o atributo de RichParallelSinkFunction:

    - subtaskIndex sobre a simultaneidade de cada operador NettySink.
    - Simultaneidade do operador NettySink.
  - RegisterServerHandler

    O RegisterServerHandler interage com o componente do RegisterServer e define as seguintes APIs:

    - **start();**: inicia o RegisterServerHandler e estabelece um contato com o RegisterServer de terceiros.
    - **createTopicNode();**: cria um nó de tópico.
    - **register();**: registra informações como endereço IP, número da porta e simultaneidade ao nó do tópico.
    - **deleteTopicNode();**: exclui um nó de tópico.
    - **unregister();**: exclui informações de registro.
    - **query();**: consulta informações de registro.
    - **isExist();**: verifica se existe uma informação específica.
    - **shutdown();**: desabilita o RegisterServerHandler e desconecta do RegisterServer de terceiros.

NOTA

- A API de RegisterServerHandler permite que o ZooKeeper funcione como o manipulador do RegisterServer. Você pode personalizar seu manipulador conforme necessário. As informações são armazenadas no ZooKeeper da seguinte forma:

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

- Informações sobre NameSpace podem ser obtidas a partir dos seguintes parâmetros do arquivo **flink-conf.yaml**:

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

- A autenticação simples da camada de autenticação e segurança (SASL) entre ZookeeperRegisterServerHandler e ZooKeeper é implementada através da estrutura de Flink.
- Certifique-se de que cada job tenha um tópico único. Caso contrário, a relação de assinatura pode não ser clara.
- Ao chamar **shutdown()**, o ZookeeperRegisterServerHandler exclui as informações de registro sobre a simultaneidade atual e tenta excluir o nó do tópico. Se o nó do tópico não estiver vazio, a exclusão será cancelada, porque nem toda a simultaneidade saiu.

- NettyServer

  NettyServer é o núcleo do operador NettySink, cuja principal função é criar um NettyServer e receber solicitações de conexão do NettyClient. Use o NettyServerHandler para enviar dados recebidos de operadores upstream de um mesmo job. O número da porta e a sub-rede do NettyServer precisam ser configurados no arquivo **flink-conf.yaml**.

  - Intervalo da porta

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

  - Sub-rede

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

    NOTA

    O parâmetro **nettyconnector.sinkserver.subnet** é definido como a sub-rede (endereço IP do serviço) do cliente de Flink por padrão. Se o cliente e o TaskManager não estiverem na mesma sub-rede, poderá ocorrer um erro. Portanto, você precisa definir manualmente esse parâmetro para a sub-rede (endereço IP do serviço) do TaskManager.

- NettyServerHandler

  O manipulador permite a interação entre NettySink e assinantes. Depois que o NettySink recebe mensagens, o manipulador envia essas mensagens. Para garantir a segurança da transmissão de dados, este canal é criptografado usando SSL. O **nettyconnector.ssl.enabled** configura se a criptografia SSL deve ser ativada. A encriptação SSL é ativada apenas quando **nettyconnector.ssl.enabled** está definido como **true**.

- **NettySource**

  O NettySource é composto pelos seguintes módulos principais:

- RichParallelSourceFunction

  NettySource herda RichParallelSinkFunction e atributos dos operadores Source. A API de RichParallelSourceFunction implementa as seguintes funções:

  - Inicia o operador NettySink.
  - Executa o operador NettySink, recebe dados de assinantes e injeta os dados em jobs.
  - Cancela a execução dos operadores Source.

  As seguintes informações podem ser obtidas usando o atributo de RichParallelSourceFunction:

  - subtaskIndex sobre a simultaneidade de cada operador NettySource.
  - Simultaneidade do operador NettySource.

  Quando o operador NettySource entra no estágio de execução, o status do NettyClient é monitorado. Quando a anormalidade ocorre, o NettyClient é reiniciado e reconectado ao NettyServer para evitar confusão de dados.

- RegisterServerHandler

  RegisterServerHandler de NettySource tem função semelhante ao RegisterServerHandler de NettySink. Obtém o endereço IP, o número da porta e as informações dos operadores concorrentes de cada job subscrito obtido no operador NettySource.

- NettyClient

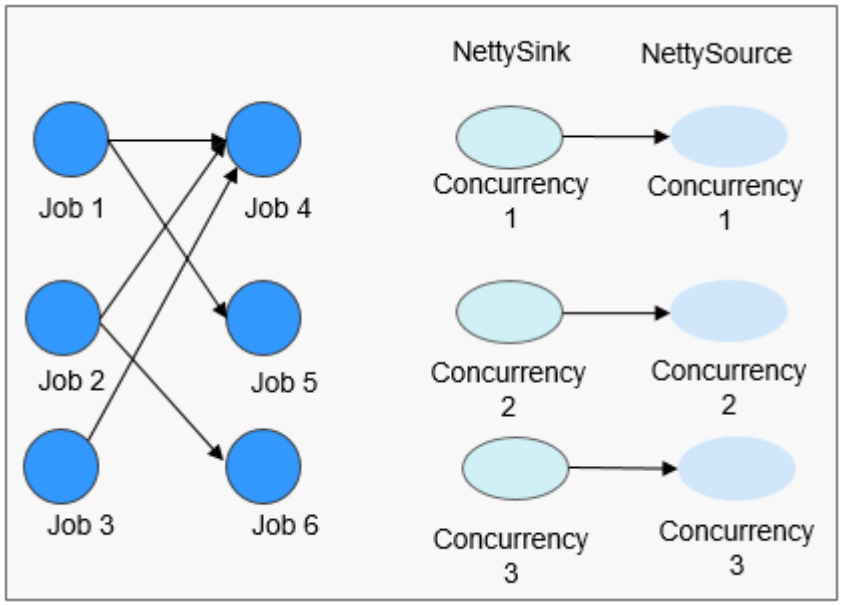
  O NettyClient estabelece uma conexão com o NettyServer e usa o NettyClientHandler para receber dados. Cada operador NettySource deve ter um nome exclusivo (especificado pelo usuário). O NettyServer determina se cada cliente vem de NettySources diferentes com base em nomes exclusivos. Quando uma conexão é estabelecida entre NettyClient e NettyServer, NettyClient é registrado com NettyServer e o nome de NettySource em NettyClient é transferido para NettyServer.

- NettyClientHandler

  O NettyClientHandler permite a interação com editores e outros operadores do job. Quando as mensagens são recebidas, o NettyClientHandler as transfere para o job. Para garantir a transmissão segura de dados, a criptografia SSL é ativada para a comunicação com o NettySink. A encriptação SSL só é ativada quando o SSL está ativado e **nettyconnector.ssl.enabled** está definido como **true**.

A relação entre os jobs pode ser muitos-para-muitos. A simultaneidade entre cada operador NettySink e NettySource é um-para-muitos, como mostrado em **Figura 6-21**.

Figura 6-21 Diagrama de relação



6.6.4.3 Stream SQL Join

Recurso de código aberto aprimorado: Stream SQL Join

Table API&SQL do Flink é uma API de consulta integrada para Scala e Java que permite a composição de consultas a partir de operadores relacionais como seleção, filtro e junção de forma intuitiva. Para detalhes sobre Table API & SQL, visite o site oficial em **https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/table/index.html**.

Introdução à Stream SQL Join

SQL Join é usada para consultar dados com base na relação entre colunas em duas ou mais tabelas. Stream SQL Join do Flink permite que você junte duas tabelas de streaming e consulte os resultados delas. Consultas semelhantes às seguintes são suportadas:

```
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;
```

Atualmente, Stream SQL Join precisa ser executada dentro de uma janela especificada. A operação de junção para dados dentro da janela requer pelo menos um predicado de junção por igualdade e uma condição de junção que limite o tempo em ambos os lados. Tal condição pode ser definida por dois predicados de intervalo apropriados (<, <=, >=, >), um predicado **BETWEEN**, ou um único predicado de igualdade que compara o mesmo tipo de atributos de tempo (como tempo de processamento ou tempo de evento) de ambas as tabelas de entrada.

O exemplo a seguir unirá todos os pedidos com seus envios correspondentes se o pedido tiver sido enviado quatro horas após o recebimento do 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. Stream SQL Join oferece suporte apenas à junção interna.
2. A cláusula **ON** deve incluir uma condição de junção igual.
3. Os atributos de tempo suportam apenas o tempo de processamento e o tempo do evento.
4. A condição de janela suporta apenas o intervalo de tempo limitado, por exemplo, **o.proctime BETWEEN s.proctime - INTERVAL '1' HOUR AND s.proctime + INTERVAL '1' HOUR**. O intervalo ilimitado, como **o. proctime > s.proctime** não é suportado. O atributo **proctime** de dois fluxos deve ser incluído. **o.proctime BETWEEN proctime () AND proctime () + 1** não é suportado.

6.6.4.4 Flink CEP em SQL

Flink CEP em SQL

O Flink permite que os usuários representem resultados de consultas de processamento de eventos complexos (CEP) em SQL para correspondência de padrões e avaliem fluxos de eventos nos mecanismos do Flink.

Sintaxe da consulta SQL

CEP SQL é implementado através da sintaxe SQL **MATCH_RECOGNIZE**. A cláusula **MATCH_RECOGNIZE** é suportada pelo Oracle SQL desde o Oracle Database 12c e é usada para indicar a correspondência de padrões de eventos em SQL. O Apache Calcite também suporta a cláusula **MATCH_RECOGNIZE**.

O Flink usa Calcite para analisar os resultados da consulta SQL. Portanto, esta operação está em conformidade com a sintaxe 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 ]*
      )
```

Os elementos de sintaxe da cláusula **MATCH_RECOGNIZE** são definidos da seguinte forma:

(Opcional) **-PARTITION BY**: define colunas de partição. Esta cláusula é facultativa. Se este parâmetro não for definido, o paralelismo 1 é usado.

(Opcional) **-ORDER BY**: define a sequência de eventos em um fluxo de dados. A cláusula **ORDER BY** é opcional. Se for ignorado, a ordenação não determinística é usada. Como a ordem dos eventos é importante na correspondência de padrões, essa cláusula deve ser especificada na maioria dos casos.

(Opcional) **-MEASURES**: especifica o valor do atributo do evento correspondido com sucesso.

(Opcional) **-ONE ROW PER MATCH | ALL ROWS PER MATCH**: define como produzir o resultado. **ONE ROW PER MATCH** indica que apenas uma linha é saída para cada correspondência. **ALL ROWS PER MATCH** indica que uma linha é saída para cada evento correspondente.

(Opcional) **-AFTER MATCH**: especifica a posição inicial para processamento após o próximo padrão ser correspondido com sucesso.

-PATTERN: define o padrão de correspondência como uma expressão regular. Os seguintes operadores podem ser usados na cláusula **PATTERN**: operadores de junção, operadores quantificadores (*, +, ?, {n}, {n,}, {n,m} e {,m}), operadores de ramificação (barra vertical |) e operadores diferenciais ('{- -}').

(Opcional) **-WITHIN**: produz uma correspondência de cláusula padrão somente quando a correspondência ocorre dentro do tempo especificado.

(Opcional) **-SUBSET**: combina uma ou mais variáveis associadas definidas na cláusula **DEFINE**.

-DEFINE: especifica a condição booleana, que define as variáveis usadas na cláusula **PATTERN**.

Além disso, a cláusula **MATCH_RECOGNIZE** suporta as seguintes funções:

-MATCH_NUMBER(): usado na cláusula **MEASURES** para alocar o mesmo número para cada linha que é correspondida com êxito.

-CLASSIFIER(): usado na cláusula **MEASURES** para indicar o mapeamento entre linhas e variáveis correspondentes.

-FIRST() e **LAST()**: usadas na cláusula **MEASURES** para retornar o valor da expressão avaliada na primeira ou na última linha do conjunto de linhas mapeado para a variável de esquema.

-NEXT() e **PREV()**: usadas na cláusula **DEFINE** para avaliar uma expressão usando a linha anterior ou a próxima em uma partição.

Palavras-chave **-RUNNING** e **FINAL**: usadas para determinar a semântica necessária para a agregação. **RUNNING** pode ser usado nas cláusulas **MEASURES** e **DEFINE**, enquanto **FINAL** pode ser usado apenas na cláusula **MEASURES**.

- Funções agregadas (**COUNT**, **SUM**, **AVG**, **MAX**, **MIN**): usadas nas cláusulas **MEASURES** e **DEFINE**.

Exemplo de consulta

A consulta a seguir encontra o padrão em forma de V no fluxo de dados do preço das ações.

```
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)
    )
```


Na consulta a seguir, a função agregada **AVG** é usada na cláusula **MEASURES** do **SUBSET E** que consiste em variáveis relacionadas a A e C.

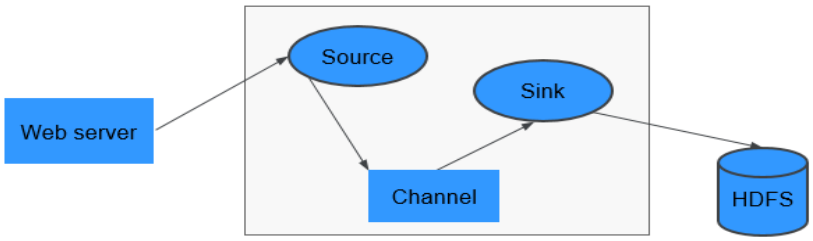
```
SELECT *
    FROM Ticker
    MATCH_RECOGNIZE (
      MEASURES
        AVG(E.price) AS avgPrice
      ONE ROW PER MATCH
      AFTER MATCH SKIP PAST LAST ROW
      PATTERN (A B+ C)
      SUBSET E = (A,C)
      DEFINE
        A AS A.price < 30,
        B AS B.price < 20,
        C AS C.price < 30
 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

6.7 Flume

6.7.1 Princípios básicos do Flume

O Flume é um sistema distribuído confiável e de HA que suporta coleta, agregação e transmissão de logs em massa. O Flume suporta a personalização de vários remetentes de dados no sistema de registro para coleta de dados. Além disso, o Flume pode processar dados e gravar dados em vários receptores de dados (personalizáveis). Flume-NG é um ramo do Flume. É simples, pequeno e fácil de implantar. A figura a seguir mostra a arquitetura básica do Flume-NG.

Figura 6-22 Arquitetura do Flume-NG



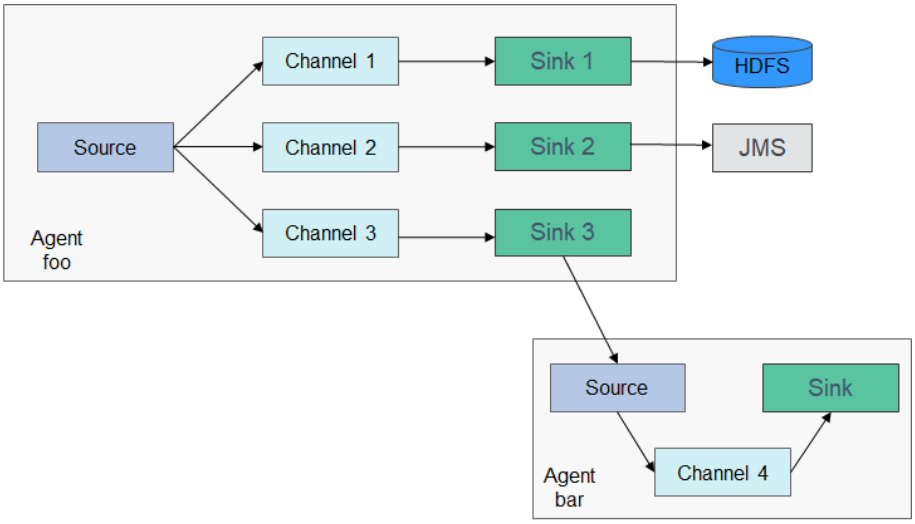
Um Flume-NG consiste em agentes. Cada agente consiste em três componentes (fonte, canal e coletor). Uma fonte é usada para receber dados. Um canal é usado para transmitir dados. Um coletor é usado para enviar dados para a próxima extremidade.

Tabela 6-5 Descrição do módulo

Módulo	Descrição
Fonte	Uma fonte recebe dados ou gera dados usando um mecanismo especial e coloca os dados em lotes em um ou mais canais. A fonte pode funcionar no modo orientado a dados ou sondagem. Os tipos de fonte típicos são os seguintes: ● Fontes que integram ao sistema, como Syslog e Netcat ● Fontes que geram eventos automaticamente, como Exec e SEQ ● IPC fontes que são usadas para comunicação entre agentes, como Avro Uma fonte deve estar associada a pelo menos um canal.
Canal	Um canal é usado para armazenar dados entre uma origem e um coletor. O canal armazena em cache os dados da fonte e exclui esses dados depois que o coletor envia os dados para o canal do próximo salto ou destino final. Diferentes canais fornecem diferentes níveis de persistência. ● Canal de memória: não persistência ● Canal de arquivo: persistência baseada em registro de gravação antecipada (WAL) ● Canal JDBC: persistência implementada com base no banco de dados embarcado O canal suporta o recurso de transação para garantir operações sequenciais simples. Um canal pode trabalhar com fontes e sumidouros de qualquer quantidade.
Coletor	Um coletor envia dados para o canal do próximo salto ou destino final. Uma vez concluído, os dados transmitidos são removidos do canal. Os tipos típicos de coletor são os seguintes: ● Coletores que enviam dados de armazenamento para o destino final, como HDFS e HBase ● Coletores que são consumidos automaticamente, como Null Sink ● IPC coletores usados para comunicação entre agentes, como o Avro Um coletor deve estar associado a um canal específico.

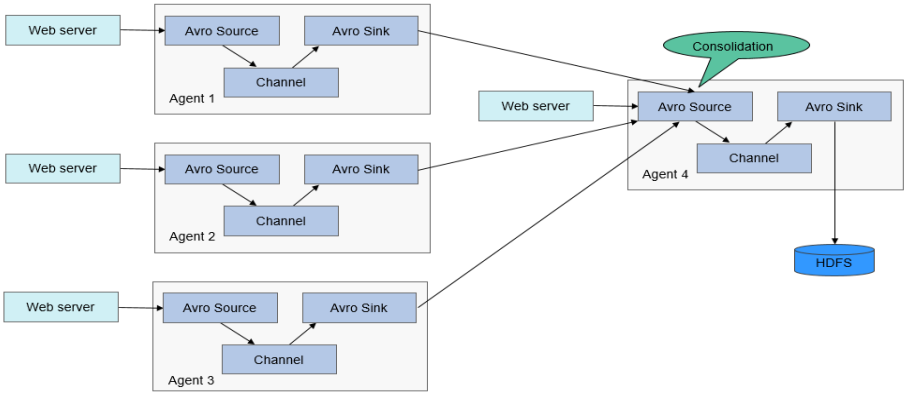
Como mostrado em **Figura 6-23**, um cliente de Flume pode ter várias fontes, canais e coletores.

Figura 6-23 Estrutura do Flume



A confiabilidade do Flume depende da troca de transações entre agentes. Se o próximo agente quebrar, o canal armazenará os dados de forma persistente e transmitirá os dados até que o agente se recupere. A disponibilidade do Flume depende dos mecanismos integrados de balanceamento de carga e failover. Tanto o canal quanto o agente podem ser configurados com várias entidades entre as quais eles podem usar políticas de balanceamento de carga. Cada agente é um processo de Máquina Virtual Java (JVM). Um servidor pode ter vários agentes. Os nós de coleta (por exemplo, Agentes 1, 2, 3) processam logs. Os nós de agregação (por exemplo, Agente 4) gravam os logs no HDFS. O agente de cada nó de coleta pode selecionar vários nós de agregação para balanceamento de carga.

Figura 6-24 Cascata de Flume



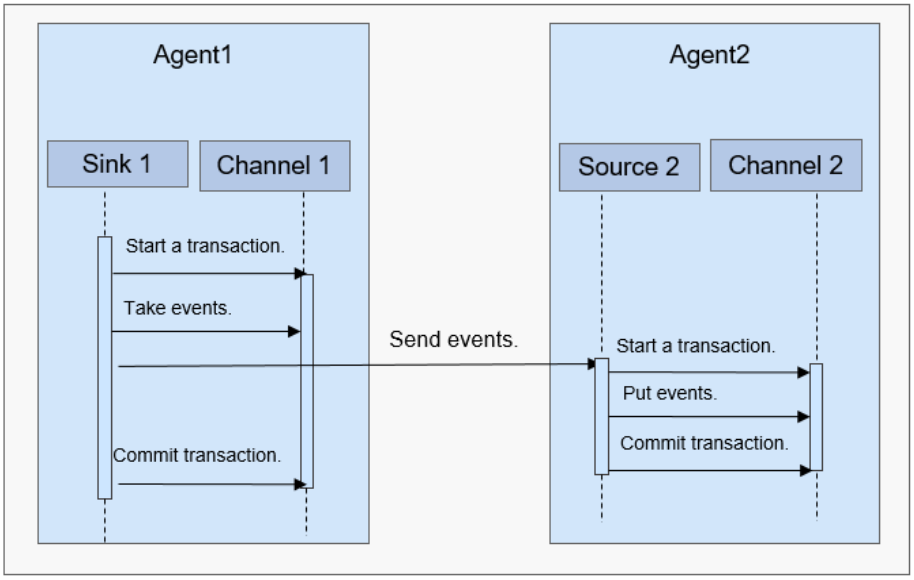
Para obter detalhes sobre a arquitetura e os princípios do Flume, consulte **https://flume.apache.org/releases/1.9.0.html**.

Princípio

Confiabilidade entre agentes

Figura 6-25 mostra a troca de dados entre os agentes.

Figura 6-25 Processo de transmissão de dados



1. O Flume garante transmissão de dados confiável com base em transações. Quando os dados fluem de um agente para outro agente, as duas transações entram em vigor. O coletor do Agente 1 (agente que envia uma mensagem) precisa obter uma mensagem de um canal e envia a mensagem para o Agente 2 (agente que recebe a mensagem). Se o Agente 2 receber e processar com sucesso a mensagem, o Agente 1 enviará uma transação, indicando uma transmissão de dados bem-sucedida e confiável.
2. Quando o Agente 2 recebe a mensagem enviada pelo Agente 1 e inicia uma nova transação, depois que os dados são processados com sucesso (gravados em um canal), o Agente 2 envia a transação e envia uma resposta bem-sucedida ao Agente 1.
3. Antes de uma operação de confirmação, se a transmissão de dados falhar, a última transcrição é iniciada e retransmite os dados que não foram transmitidos da última vez. A operação de commit gravou a transação em um disco. Portanto, a última transação pode continuar após o processo falhar e restaurar.

6.7.2 Relação entre Flume e outros componentes

Relação entre Flume e HDFS

Se o HDFS estiver configurado como o coletor do Flume, o HDFS funcionará como o sistema de armazenamento de dados final do Flume. O Flume instala, configura e grava todos os dados transmitidos no HDFS.

Para obter detalhes, consulte **Cenário típico: coleção de logs estáticos locais e carregamento para HDFS** e **Cenário típico: coleção de logs estáticos locais e carregamento para HDFS**.

Relação entre Flume e HBase

Se o HBase estiver configurado como o coletor do Flume, o HBase funcionará como o sistema de armazenamento de dados final do Flume. O Flume grava todos os dados transmitidos no HBase com base nas configurações. Para obter detalhes, consulte **Cenário típico: coleção de logs estáticos locais e carregamento para HBase**.

6.7.3 Recursos de código aberto aprimorados do Flume

Recursos de código aberto aprimorados do Flume

- Melhorando a velocidade de transmissão: várias linhas em vez de apenas uma linha de dados podem ser especificadas como um evento. Isso melhora a eficiência da execução de código e reduz o tempo de gravação em disco.
- Transferindo arquivos binários ultra-grandes: de acordo com o uso de memória atual, o Flume ajusta automaticamente a memória usada para transferir arquivos binários ultra-grandes para evitar falta de memória.
- Apoiando a personalização de preparações antes e depois da transmissão: o Flume suporta scripts personalizados para serem executados antes ou depois da transmissão para fazer preparações.
- Gerenciando alarmes do cliente: o Flume recebe os alarmes do cliente através do MonitorServer e relata os alarmes ao centro de gerenciamento de alarmes no MRS Manager.

6.8 HBase

6.8.1 Princípios básicos do HBase

O HBase realiza o armazenamento de dados. O HBase é um sistema de armazenamento distribuído de código aberto, orientado a colunas, adequado para armazenar grandes quantidades de dados não estruturados ou semi estruturados. Possui alta confiabilidade, alto desempenho e escalabilidade flexível e suporta leitura/gravação de dados em tempo real. Para obter mais informações sobre o HBase, **https://hbase.apache.org/**.

As características típicas de uma tabela armazenada no HBase são as seguintes:

- Tabela grande (BigTable): uma tabela contém centenas de milhões de linhas e milhões de colunas.
- Orientado à coluna: armazenamento orientado a colunas, recuperação e controle de permissão
- Esparso: colunas nulas na tabela não ocupam nenhum espaço de armazenamento.

O MRS HBase suporta indexação secundária para permitir que índices sejam criados para valores de coluna, de modo que os dados possam ser filtrados por coluna usando APIs do HBase nativo.

Arquitetura do HBase

Um cluster do HBase consiste em processos de HMaster ativo e em espera e vários processos do RegionServer.

Figura 6-26 Arquitetura do HBase

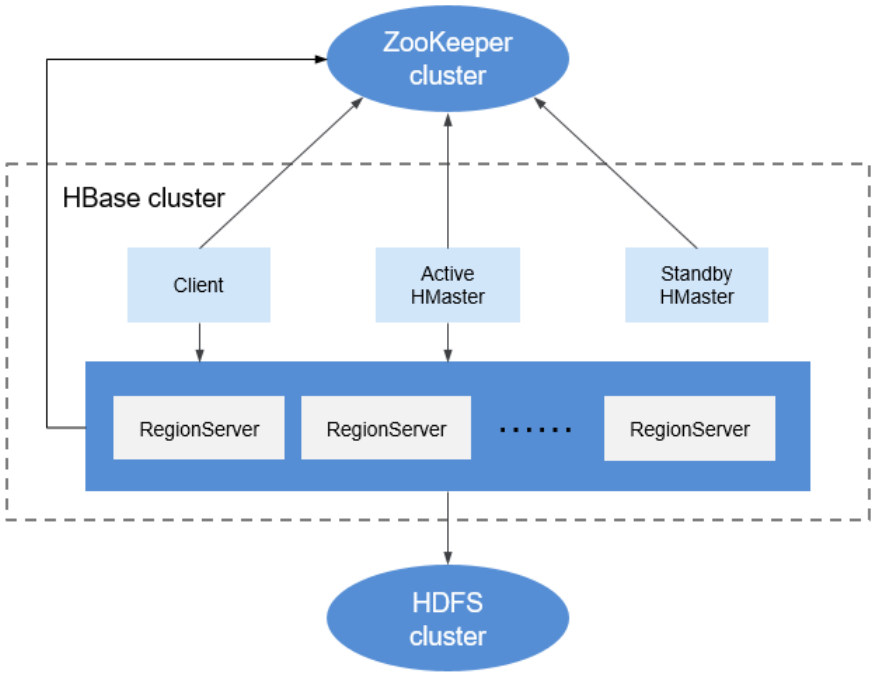


Tabela 6-6 Descrição do módulo

Módulo	Descrição
Master	Master também é chamado de HMaster. No modo HA, o HMaster consiste em um HMaster ativo e um HMaster em espera. ● Master ativo: gerencia RegionServer no HBase, incluindo a criação, exclusão, modificação e consulta de uma tabela, equilibra a carga de RegionServer, ajusta a distribuição de Region, divide e distribui Region depois de dividida e migra Region após RegionServer expirar. ● Master em espera: assume os serviços quando o HMaster ativo está com defeito. O HMaster ativo original é rebaixado para o HMaster em espera depois que a falha é corrigida.
Client	Client se comunica com o Master para gerenciamento e com o RegionServer para proteção de dados usando Chamada de procedimento remoto mecanismo (RPC) do HBase.
RegionServer	O RegionServer fornece serviços de leitura e gravação de dados de tabela como uma unidade de processamento e computação de dados no HBase. O RegionServer é implementado com DataNodes de clusters do HDFS para armazenar dados.
Cluster do ZooKeeper	O ZooKeeper fornece serviços de coordenação distribuídos para processos em clusters do HBase. Cada RegionServer é registrado no ZooKeeper para que o Mestre ativo possa obter o status de saúde de cada RegionServer.

Módulo	Descrição
Cluster do HDFS	O HDFS fornece serviços de armazenamento de arquivos altamente confiáveis para o HBase. Todos os dados do HBase são armazenados no HDFS.

Princípios do HBase

- **Modelo de dados do HBase**

  O HBase armazena dados em tabelas, conforme mostrado na **Figura 6-27**. Os dados em uma tabela são divididos em várias Regions, que são alocadas pelo Master para RegionServers para gerenciamento.

  Cada Region contém dados dentro de um intervalo de RowKey. Uma tabela de dados do HBase contém apenas uma região no início. À medida que o número de dados aumenta e atinge o limite superior da capacidade da Region, a Region é dividida em duas Regiões. Você pode definir o intervalo de RowKey de uma Region ao criar uma tabela ou definir o tamanho da Region no arquivo de configuração.

  Figura 6-27 Modelo de dados do HBase

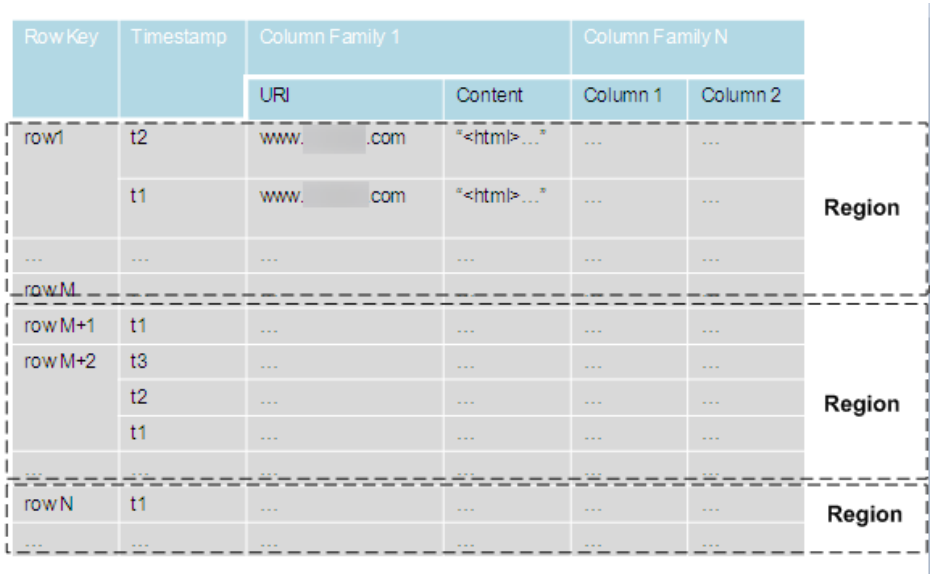


Tabela 6-7 Conceitos

Módulo	Descrição
RowKey	Semelhante à chave primária em uma tabela de relacionamento, que é o ID exclusivo dos dados em cada linha. Uma RowKey pode ser uma cadeia, inteiro ou binário. Todos os registros são armazenados depois de serem classificados por RowKey.
Timestamp	O timestamp de uma operação de dados. Os dados podem ser especificados com versões diferentes por carimbo de data/hora. Os dados de diferentes versões em cada cell são armazenados por tempo em ordem decrescente.

Módulo	Descrição
Cell	Unidade mínima de armazenamento de HBase, consistindo de chaves e valores. Uma chave consiste em seis campos, a saber namely row, column family, column qualifier, timestamp, type e MVCC version. Os valores são os objetos de dados binários.
Column Family	Uma ou várias column families horizontais formam uma tabela. Uma column family pode consistir em várias columns aleatórias. Uma column é um rótulo sob uma column family, que pode ser adicionado conforme necessário quando os dados são gravados. A column family suporta expansão dinâmica para que o número e o tipo de columns não precisem ser predefinidos. Columns de uma tabela no HBase são esparsamente distribuídas. O número e o tipo de columns em linhas diferentes podem ser diferentes. Cada column family tem o tempo de vida independente (TTL). Você pode bloquear apenas a linha. As operações na linha de uma column family são as mesmas que as de outras linhas.
Column	Semelhante aos bancos de dados tradicionais, as tabelas HBase também usam columns para armazenar dados do mesmo tipo.

- **Armazenamento de dados do RegionServer**

  O RegionServer gerencia as regiões alocadas pelo HMaster. **Figura 6-28** mostra a estrutura de armazenamento de dados do RegionServer.

  Figura 6-28 Estrutura de armazenamento de dados RegionServer

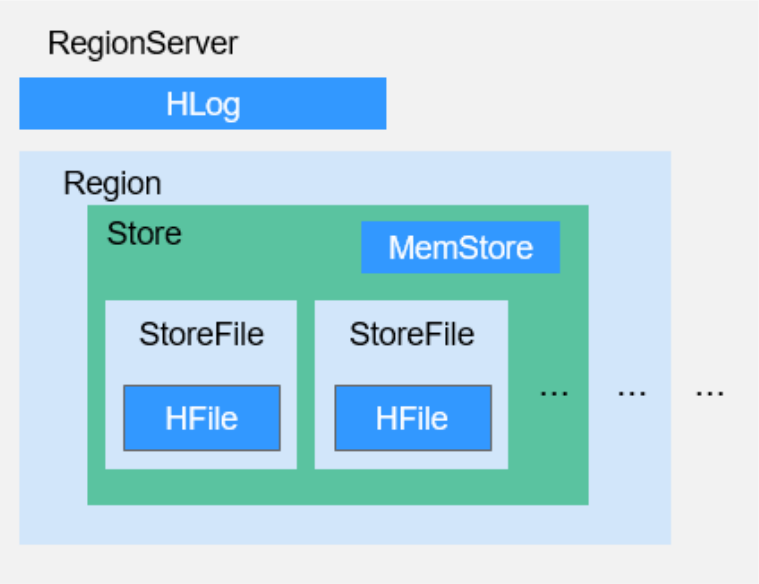


  Tabela 6-8 lista cada componente da Region descrita em **Figura 6-28**.

Tabela 6-8 Descrição da estrutura da Region

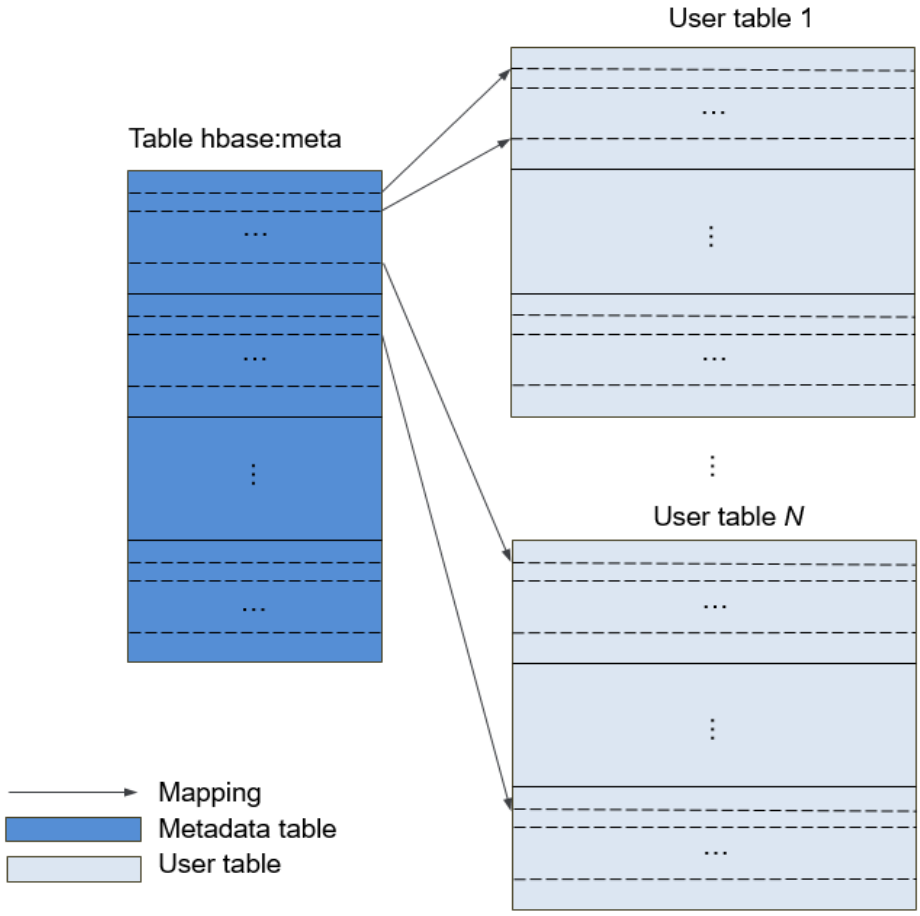
Módulo	Descrição
Store	Uma Region consiste em um ou vários Stores. Cada Store mapeia uma column family em **Figura 6-27**.
MemStore	Um Store contém um MemStore. O MemStore armazena em cache os dados inseridos em uma Region pelo cliente. Quando a capacidade do MemStore atinge o limite superior, o RegionServer libera os dados do MemStore para o HDFS.
StoreFile	Os dados liberados no HDFS são armazenados como um StoreFile no HDFS. À medida que mais dados são inseridos, vários StoreFiles são gerados em um Store. Quando o número de StoreFiles atinge o limite superior, RegionServer mescla vários StoreFiles em um StoreFile grande.
HFile	HFile define o formato de armazenamento de StoreFiles em um sistema de arquivos. HFile é a implementação subjacente do StoreFile.
HLog	HLogs evitam a perda de dados quando o RegionServer está com defeito. Várias Regions em um RegionServer compartilham o mesmo HLog.

- **Tabela de metadados**

  A tabela de metadados é uma tabela do HBase especial, que é usada pelo cliente para localizar uma região. A tabela de metadados inclui **hbase:meta** para registrar informações de região de tabelas de usuário, como o local da região e início e fim de RowKey.

  Figura 6-29 mostra a relação de mapeamento entre tabelas de metadados e tabelas de usuário.

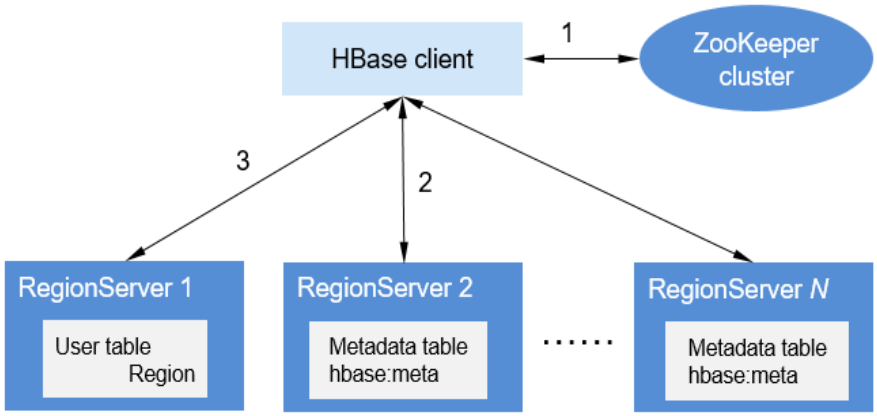
Figura 6-29 Relações de mapeamento entre tabelas de metadados e tabelas de usuário



- **Processo de operação de dados**

  Figura 6-30 mostra o processo de operação de dados do HBase.

Figura 6-30 Processamento de dados



a. Quando você adiciona, exclui, modifica e consulta dados do HBase, o cliente do HBase primeiro se conecta a ZooKeeper para obter informações sobre o RegionServer onde a tabela **hbase:meta** está localizada. Se você modificar o

namespace, como criar e excluir uma tabela, precisará acessar o HMaster para atualizar as meta informações.

b. O cliente do HBase se conecta à RegionServer onde a região da tabela **hbase:meta** está localizada e obtém o local RegionServer onde a região da tabela de usuário reside.

c. Em seguida, o cliente do HBase se conecta à RegionServer onde a região da tabela de usuários está localizada e emite um comando de operação de dados para o RegionServer. O RegionServer executa o comando.

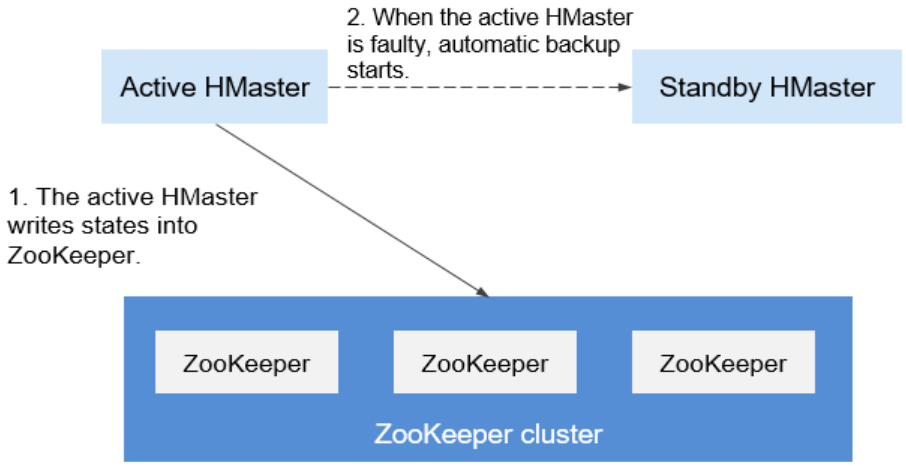
Para melhorar a eficiência do processamento de dados, o cliente HBase armazena em cache as informações de região da tabela **hbase:meta** e da tabela de usuário. Quando uma aplicação inicia uma segunda operação de dados, o cliente do HBase consulta as informações de região da memória. Se nenhuma correspondência for encontrada na memória, o cliente do HBase executará as operações anteriores para obter informações de região.

6.8.2 Solução HA do HBase

HA do HBase

HMaster no HBase aloca Regiões. Quando um serviço do RegionServer é interrompido, o HMaster migra a Região correspondente para outro RegionServer. O recurso HA do HMaster é trazido para impedir que as funções do HBase sejam afetadas pelo ponto único de falha (SPOF) do HMaster.

Figura 6-31 Arquitetura de implementação HA do HMaster



A arquitetura HA do HMaster é implementada criando o nó efêmero do ZooKeeper em um cluster do ZooKeeper.

Na inicialização, os nós do HMaster tentam criar um znode principal no cluster do ZooKeeper. O nó HMaster que cria o znode principal primeiro se torna o HMaster ativo e o outro é o HMaster em espera.

Ele adicionará eventos de relógio ao nó principal. Se o serviço no HMaster ativo for interrompido, o HMaster ativo será desconectado do cluster do ZooKeeper. Depois que a sessão expira, o HMaster ativo desaparece. O HMaster em espera detecta o desaparecimento

do HMaster ativo por meio de eventos de observação e cria um nó principal para tornar-se o ativo. Em seguida, a alternância ativa/em espera é concluída. Se o nó com falha detectar a existência do nó principal após ser reiniciado, ele entrará no estado de espera e adicionará eventos de observação ao nó principal.

Quando o cliente acessa o HBase, ele primeiro obtém o endereço do HMaster com base nas informações do nó principal no ZooKeeper e estabelece uma conexão com o HMaster ativo.

6.8.3 Relação com outros componentes

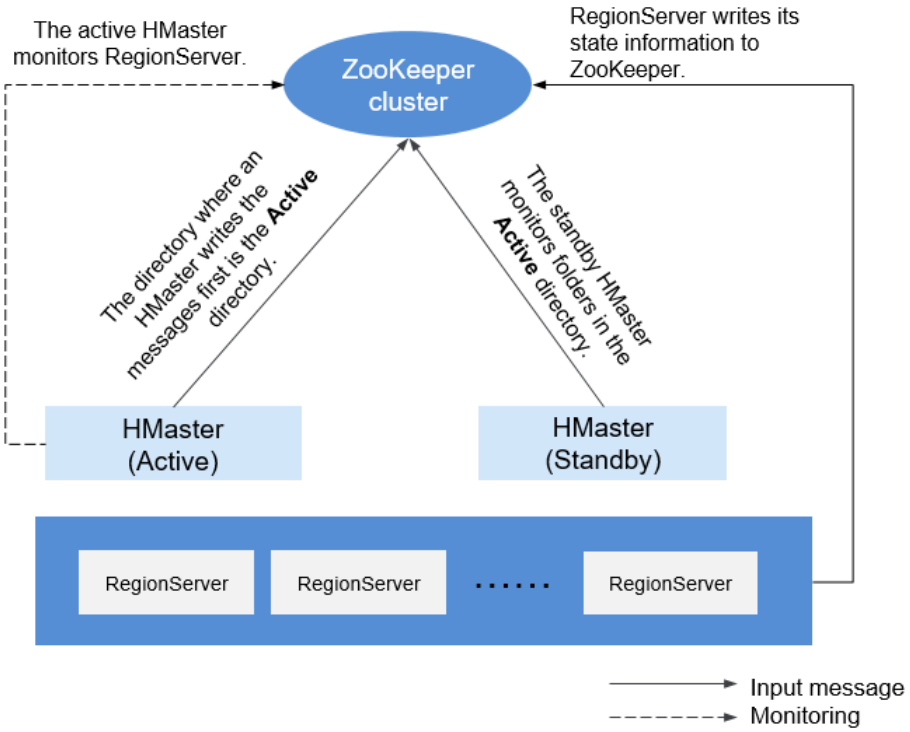
Relação entre HDFS e HBase

HDFS é o subprojeto do Apache Hadoop. O HBase usa o sistema de arquivos distribuídos do Hadoop (HDFS) como sistema de armazenamento de arquivos. O HBase está localizado na camada de armazenamento estruturada. O HDFS fornece suporte altamente confiável para armazenamento de camada inferior do HBase. Todos os arquivos de dados do HBase podem ser armazenados no HDFS, exceto alguns arquivos de log gerados pelo HBase.

Relação entre ZooKeeper e HBase

Figura 6-32 descreve a relação entre o ZooKeeper e o HBase.

Figura 6-32 Relação entre ZooKeeper e HBase



1. O HRegionServer registra-se em ZooKeeper no nó efêmero. O ZooKeeper armazena as informações do HBase, incluindo os metadados do HBase e os endereços de HMaster.
2. HMaster detecta o status de saúde de cada HRegionServer usando o ZooKeeper e os monitora.
3. O HBase pode implementar vários HMasters (como NameNode do HDFS). Quando o nó HMaster ativo está com defeito, o nó HMaster em espera obtém as informações de

estado de todo o cluster usando ZooKeeper, o que significa que as falhas de ponto único do HBase podem ser evitadas usando o ZooKeeper.

6.8.4 Recursos de código aberto aprimorados do HDFS

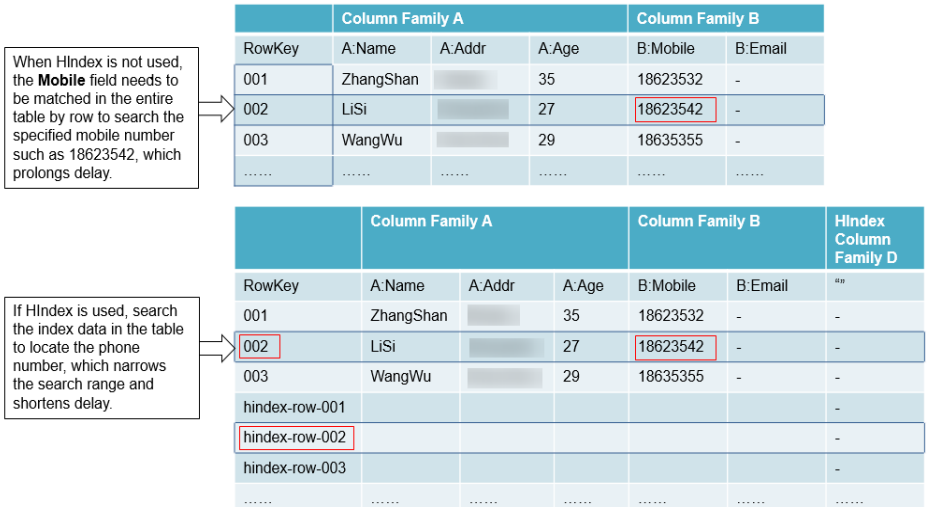
HIndex

HBase é um banco de dados de armazenamento distribuído do tipo chave-valor. Os dados de uma tabela são classificados em ordem alfabética com base em chaves de linha. Se você consultar dados com base em uma chave de linha especificada ou verificar dados na escala de uma chave de linha especificada, o HBase poderá localizar rapidamente os dados de destino, aumentando a eficiência.

No entanto, na maioria dos cenários reais, você precisa consultar os dados dos quais o valor da coluna é *XXX*. O HBase fornece o recurso Filtro para consultar dados com um valor de coluna específico. Todos os dados são verificados na ordem das chaves de linha e, em seguida, os dados são combinados com o valor da coluna específica até que os dados necessários sejam encontrados. O recurso Filtro verifica alguns dados desnecessários para obter os únicos dados necessários. Portanto, o recurso Filtro não pode atender aos requisitos de consultas frequentes com padrões de alto desempenho.

O HBase HIndex foi projetado para resolver esses problemas. O HBase HIndex permite que o HBase consulte dados com base em valores de coluna específicos.

Figura 6-33 HIndex



- Atualização contínua não é suportada para dados de índice.
- Restrições de índices combinados:
  - Todas as colunas envolvidas em índices combinados devem ser inseridas ou excluídas em uma única mutação. Caso contrário, ocorrerá inconsistência.

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

    Operações de escrita corretas:

    ```
    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);
```

Operações de escrita incorretas:

```
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);
```

- A consulta combinada baseada em condições é suportada somente quando a coluna de índice combinada contém critérios de filtro ou StartRow e StopRow não são especificados para algumas colunas de índice.

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

  Operações de consulta corretas:

```
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'}
```

  Operações de consulta incorretas:

```
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' }
```

- Não configure explicitamente nenhuma política de divisão para tabelas com dados de índice.
- Outras operações de mutação, como **increment** e **append**, não são suportadas.
- O índice da coluna com **maxVersions** maior que 1 não é suportado.
- A coluna de índice de dados em uma linha não pode ser atualizada.

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

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

Operações de atualização corretas:

```
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);
```

Operações de atualização incorretas:

```
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);
```

- A tabela à qual um índice é adicionado não pode conter um valor maior que 32 KB.
- Se os dados do usuário forem excluídos devido à expiração do TTL no nível da coluna, os dados de índice correspondentes não serão excluídos imediatamente. Eles serão excluídos na operação de compactação principal.
- O TTL da família de colunas do usuário não pode ser modificado após a criação do índice.
  - Se o TTL de uma família de colunas aumenta depois que um índice é criado, exclua o índice e recrie um. Caso contrário, alguns dados de índice gerados serão excluídos antes que os dados do usuário sejam excluídos.
  - Se o valor do TTL da família de colunas diminuir após a criação de um índice, os dados do índice serão excluídos após os dados do usuário serem excluídos.
- A consulta de índice não suporta a operação inversa e os resultados da consulta são desordenados.
- O índice não suporta a operação **clone snapshot**.
- A tabela de índice deve usar HIndexWALPlayer para reproduzir logs. WALPlayer não pode ser usado para reproduzir logs.


```
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
```

- Quando o comando **deleteall** é executado para a tabela de índice, o desempenho é baixo.
- A tabela de índice não suporta HBCK. Para usar o HBCK para reparar a tabela de índice, exclua os dados de índice primeiro.

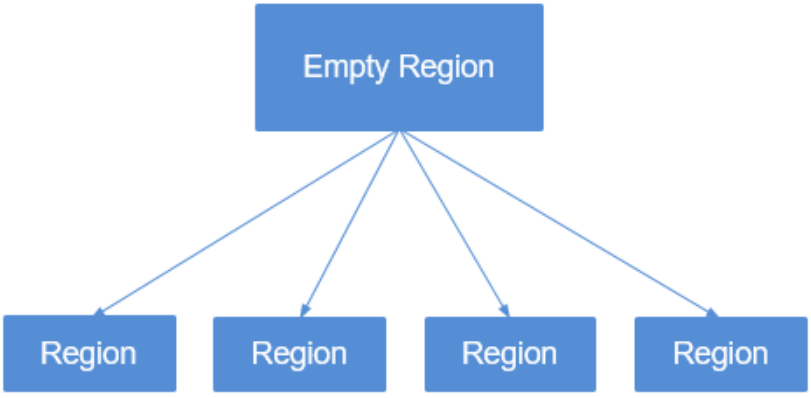
Divisão multi-ponto

Quando você cria tabelas que são pré-divididas por região no HBase, talvez você não saiba a tendência de distribuição de dados, portanto, a divisão por região pode ser inadequada. Depois que o sistema funciona por um período, as regiões precisam ser divididas novamente para obter um melhor desempenho. Somente regiões vazias podem ser divididas.

A função de divisão de região fornecida com o HBase divide regiões somente quando elas atingem o limite. Isso é chamado de "divisão de ponto único".

Para alcançar um melhor desempenho quando as regiões são divididas com base nos requisitos do usuário, é desenvolvida a divisão multi-ponto, que também é chamada de "divisão dinâmica". Ou seja, uma região vazia é pré-dividida em várias regiões para evitar a deterioração do desempenho causada por espaço de região insuficiente.

Figura 6-34 Divisão multi-ponto



Limitação de conexão

Muitas sessões significam que muitas consultas e tarefas de MapReduce estão sendo executadas no HBase, o que compromete o desempenho do HBase e até causa rejeição de serviço. Você pode configurar parâmetros para limitar o número máximo de sessões que podem ser estabelecidas entre o cliente e o servidor HBase para obter a proteção contra sobrecarga do HBase.

Recuperação de desastres aprimorada

Os recursos de recuperação de desastres (DR) entre os clusters ativo e em espera podem aprimorar o HA dos dados do HBase. O cluster ativo fornece serviços de dados e o cluster em espera faz backup dos dados. Se o cluster ativo estiver com defeito, o cluster em espera assumirá os serviços de dados. Em comparação com a função de replicação de código aberto, essa função é aprimorada da seguinte maneira:

1. A função de lista branca de cluster em espera só é aplicável ao envio de dados para um endereço IP de cluster especificado.
2. Na versão de código aberto, a replicação é sincronizada com base no WAL e o backup de dados é implementado repetindo o WAL no cluster em espera. Para operações de BulkLoad, como nenhum WAL é gerado, os dados não serão replicados para o cluster em espera. Ao registrar as operações do BulkLoad no WAL e sincronizá-las com o cluster em espera, o cluster em espera pode ler registros de operação do BulkLoad por meio do WAL e carregar o HFile no cluster ativo para implementar o backup de dados.
3. Na versão de código aberto, o HBase filtra as ACLs. Portanto, as informações de ACL não serão sincronizadas com o cluster em espera. Adicionando um filtro (**org.apache.hadoop.hbase.replication.SystemTableWALEntryFilterAllowACL**), as informações da ACL podem ser sincronizadas com o cluster de espera. Você pode configurar **hbase.replication.filter.sytemWALEntryFilter** para habilitar o filtro e implementar a sincronização da ACL.
4. Quanto à restrição somente leitura do cluster em espera, somente superusuários dentro do cluster em espera podem modificar o HBase do cluster em espera. Em outras palavras, os clientes HBase fora do cluster em espera só podem ler o HBase do cluster em espera.

HBase MOB

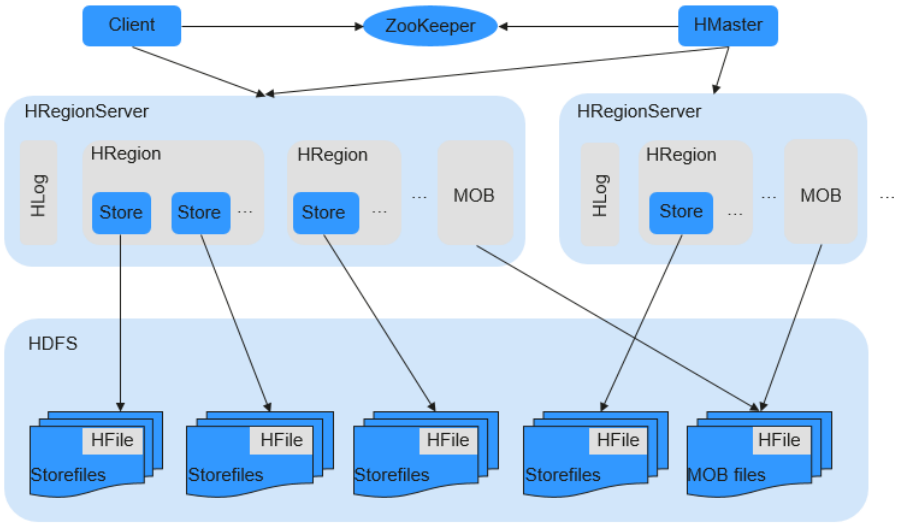
Nos cenários reais da aplicação, os dados em vários tamanhos precisam ser armazenados, por exemplo, dados de imagem e documentos. Dados cujo tamanho seja menor que 10 MB podem ser armazenados no HBase. O HBase pode produzir o melhor desempenho de leitura e gravação para dados cujo tamanho é menor que 100 KB. Se o tamanho dos dados armazenados no HBase for maior que 100 KB ou mesmo atingir 10 MB e o mesmo número de arquivos de dados for inserido, a quantidade total de dados será grande, causando compactação e divisão frequentes, alto consumo de CPU, alta frequência de I/O do disco e baixo desempenho.

Os dados MOB (cujo tamanho varia de 100 KB a 10 MB) são armazenados em um sistema de arquivos (por exemplo, HDFS) no formato HFile. As ferramentas expiradasMobFileCleaner e Sweeper são usadas para gerenciar HFiles e salvar as informações de endereço e tamanho sobre os HFiles no armazenamento do HBase como valores. Isso diminui muito a compactação e a frequência de divisão no HBase e melhora o desempenho.

Como mostrado em **Figura 6-35**, MOB indica mobstore armazenado em HRegion. O Mobstore armazena chaves e valores. Em que, uma chave é a chave correspondente no HBase

e um valor é o endereço de referência e o deslocamento de dados armazenados no sistema de arquivos. Ao ler dados, o mobstore usa seu próprio verificador para ler objetos de dados chave-valor e usa as informações de endereço e tamanho de dados no valor para obter dados de destino do sistema de arquivos.

Figura 6-35 Princípio de armazenamento de dados MOB



HFS

HBase FileStream (HFS) é um módulo independente de armazenamento de arquivos HBase. Ele é usado em aplicações de camada superior do MRS encapsulando interfaces HBase e HDFS para fornecer essas aplicações de camada superior com funções como armazenamento de arquivos, leitura e exclusão.

No ecossistema Hadoop, o HDFS e o HBase enfrentam problemas difíceis no armazenamento de arquivos em massa em alguns cenários:

- Se um grande número de arquivos pequenos forem armazenados no HDFS, o NameNode estará sob grande pressão.
- Alguns arquivos grandes não podem ser armazenados diretamente no HBase devido às APIs e mecanismos internos do HBase.

O HFS é desenvolvido para o armazenamento misto de pequenos arquivos massivos e alguns arquivos grandes no Hadoop. Simplificando, grandes arquivos pequenos (menores que 10 MB) e alguns arquivos grandes (maiores que 10 MB) precisam ser armazenados em tabelas do HBase.

Para esse cenário, o HFS fornece APIs de operação unificadas semelhantes às APIs de função do HBase.

Vários RegionServers implementados no mesmo servidor

Vários RegionServers podem ser implementados em um nó para melhorar a utilização de recursos do HBase.

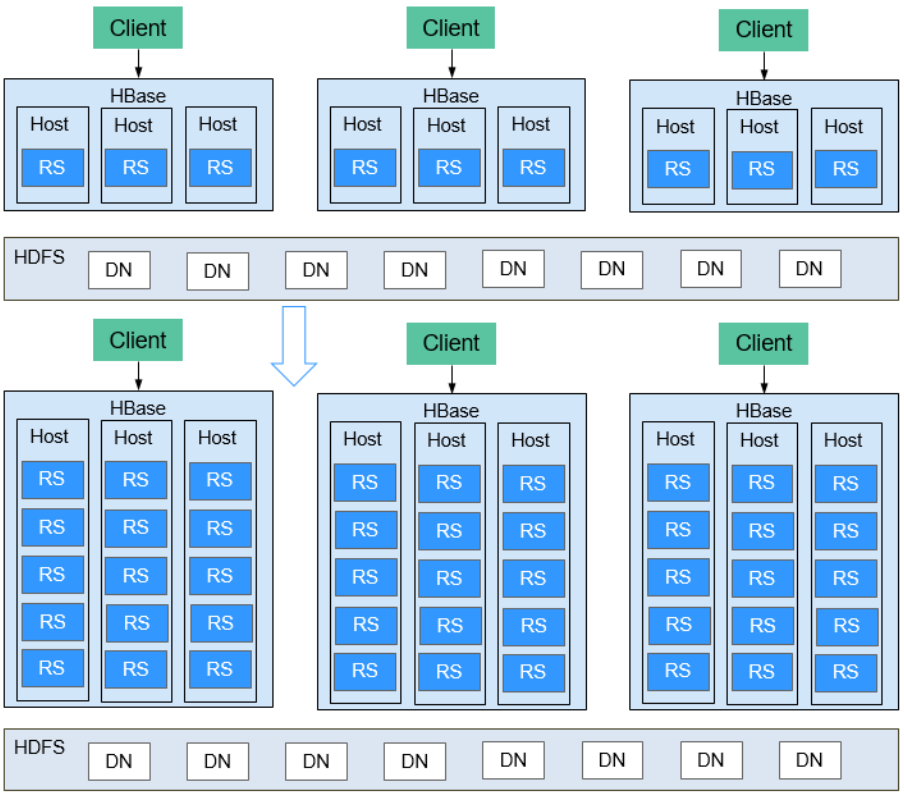
Se apenas um RegionServer for implementado, a utilização de recursos será baixa devido aos seguintes motivos:

1. Um RegionServer oferece suporte a um número limitado de regiões e, portanto, os recursos de memória e CPU não podem ser totalmente usados.
2. Um único RegionServer oferece suporte a no máximo 20 TB de dados, dos quais duas cópias exigem 40 TB e três cópias exigem 60 TB. Neste caso, a capacidade de 96 TB não pode ser utilizada.
3. Desempenho de gravação ruim: um RegionServer é implementado em um servidor físico e existe apenas um HLog. Apenas três discos podem ser gravados ao mesmo tempo.

A utilização de recursos do HBase pode ser aprimorada quando vários RegionServers são implementados no mesmo servidor.

1. Um servidor físico pode ser configurado com um máximo de cinco RegionServers. O número de RegionServers implementados em cada servidor físico pode ser configurado conforme necessário.
2. Recursos como memória, discos e CPUs podem ser totalmente utilizados.
3. Um servidor físico suporta um máximo de cinco HLogs e permite que os dados sejam gravados em 15 discos ao mesmo tempo, melhorando significativamente o desempenho de gravação.

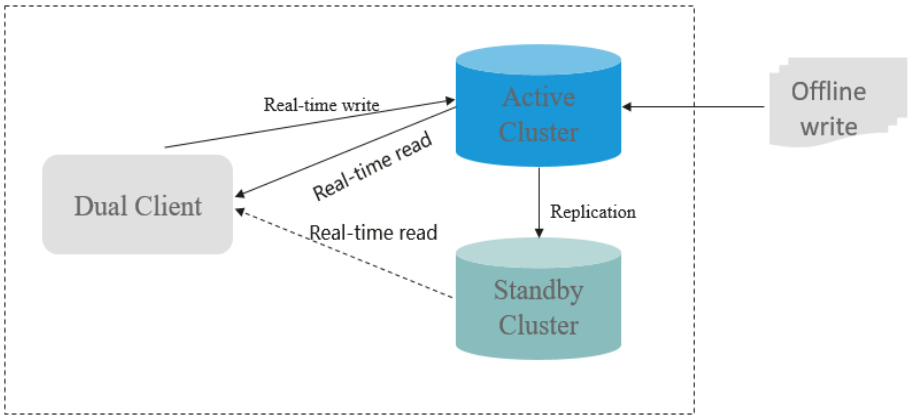
Figura 6-36 Utilização aprimorada de recursos do HBase



Leitura dupla do HBase

No cenário de armazenamento HBase, é difícil garantir 99,9% de estabilidade de consulta devido a GC, tremulação de rede e setores defeituosos de discos. O recurso de leitura dupla do HBase é adicionado para atender aos requisitos de falhas baixas durante a leitura aleatória de grande volume de dados.

O recurso de leitura dupla do HBase baseia-se na capacidade de DR dos clusters ativos e em espera. A probabilidade de que os dois aglomerados gerem falhas ao mesmo tempo, é muito menor do que a de um aglomerado. O modo de acesso simultâneo de cluster duplo é usado para garantir a estabilidade da consulta. Quando um usuário inicia uma solicitação de consulta, o serviço HBase dos dois clusters é consultado ao mesmo tempo. Se o cluster ativo não retornar nenhum resultado após um período de tempo (o tempo máximo de falha tolerável), os dados do cluster com a resposta mais rápida podem ser usados. A figura a seguir mostra o princípio de funcionamento.



6.9 HDFS

6.9.1 Princípios básicos do HDFS

O Hadoop Distributed File System (HDFS) implementa leitura/gravação confiável e distribuída de grandes quantidades de dados. O HDFS é aplicável ao cenário em que os recursos de leitura/gravação de dados "gravar uma vez e ler várias vezes". No entanto, a operação de gravação é executada em sequência, ou seja, é uma operação de gravação realizada durante a criação do arquivo ou uma operação de adição realizada por trás do arquivo existente. O HDFS garante que apenas um chamador possa executar a operação de gravação em um arquivo, mas vários chamadores possam executar a operação de leitura no arquivo ao mesmo tempo.

Arquitetura

O HDFS consiste em NameNodes ativos e em espera e vários DataNodes como mostrado em **Figura 6-37**.

O HDFS funciona em arquitetura principal/secundário. O NameNodes é executado no nó principal (ativo) e o DataNodes é executado no nó secundário (em espera). O ZKFC deve ser executado junto com o NameNodes.

A comunicação entre NameNodes e DataNodes é baseada no Protocolo de Controle de Transmissão (TCP)/Protocolo de Internet (IP). O NameNode, DataNode, ZKFC e JournalNode podem ser implantados em servidores do Linux.

Figura 6-37 Arquitetura do HDFS de HA

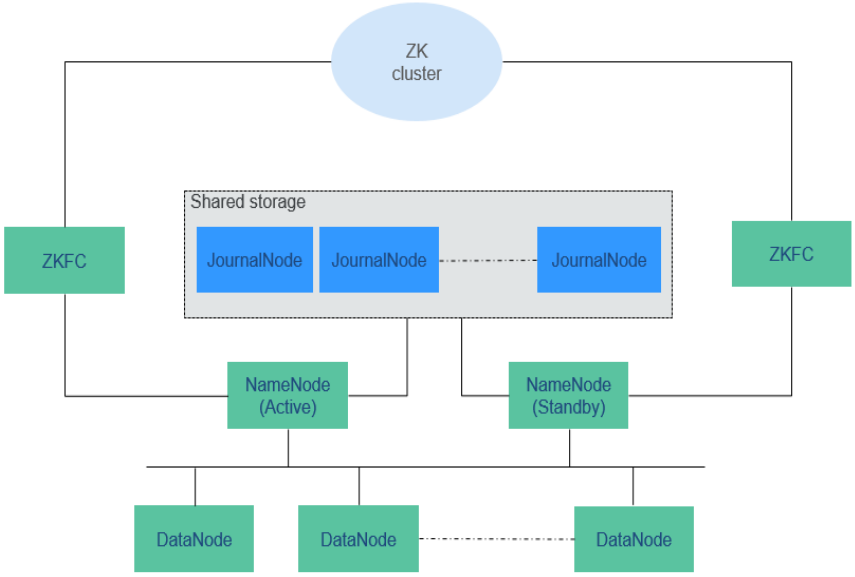


Tabela 6-9 descreve as funções de cada módulo mostrado em **Figura 6-37**.

Tabela 6-9 Descrição de módulo

Módulo	Descrição
NameNode	Um NameNode é usado para gerenciar o namespace, estrutura de diretórios e informações de metadados de um sistema de arquivos e fornecer o mecanismo de backup. O NameNode é classificado nos dois tipos seguintes: ● NameNode ativo: gerencia o namespace, mantém a estrutura de diretórios e metadados dos sistemas de arquivos e registra as relações de mapeamento entre blocos de dados e arquivos aos quais os blocos de dados pertencem. ● NameNode em espera: sincroniza com os dados no NameNode ativo e assume os serviços do NameNode ativo quando o NameNode ativo está com defeito. ● NameNode de observador: sincroniza com os dados no NameNode ativo e processa solicitações de leitura do cliente.
DataNode	Um DataNode é usado para armazenar blocos de dados de cada arquivo e relatar periodicamente o status de armazenamento para o NameNode.
JournalNode	No cluster de HA, sincroniza metadados entre os NameNodes ativos e em espera.
ZKFC	ZKFC deve ser implantado para cada NameNode. Ele monitora o status do NameNode e grava informações de status em ZooKeeper. ZKFC também tem permissões para selecionar o NameNode ativo.
ZK Cluster	ZooKeeper é um serviço de coordenação que ajuda o ZKFC a eleger o NameNode ativo.

Módulo	Descrição
HttpFS gateway	HttpFS é um processo de gateway sem estado único que fornece a API REST WebHDFS para processos externos e a API de FileSystem para o HDFS. HttpFS é usado para transmissão de dados entre diferentes versões do Hadoop. Ele também é usado como um gateway para acessar o HDFS por trás de um firewall.

- **Arquitetura do HDFS de HA**

  HA é usado para resolver o problema SPOF do NameNode. Este recurso fornece um NameNode em espera para o NameNode ativo. Quando o NameNode ativo está com defeito, o NameNode em espera pode rapidamente assumir o fornecimento contínuo de serviços para sistemas externos.

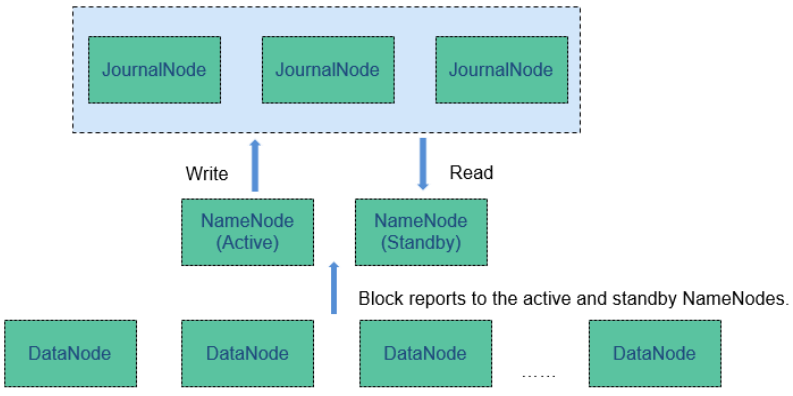
  Em um cenário típico de HA do HDFS, geralmente há dois NameNodes. Um está no estado ativo e o outro no estado de espera.

  Um sistema de armazenamento compartilhado é necessário para suportar a sincronização de metadados dos NameNodes ativa e em espera. Esta versão fornece a solução de HA do Quorum Journal Manager (QJM), conforme mostrado em **Figura 6-38**. Um grupo de JournalNodes é usado para sincronizar metadados entre os NameNodes ativos e em espera.

  Geralmente, um número ímpar (2N+1) de JournalNodes é configurado e pelo menos três JournalNodes são necessários. Para uma mensagem de atualização de metadados, a gravação de dados é considerada bem-sucedida desde que a gravação de dados seja bem-sucedida no JournalNodes N+1. Neste caso, é permitida uma falha de gravação de dados de no máximo N JournalNodes. Por exemplo, quando há três JournalNodes, a falha de gravação de dados de um JournalNode é permitida; quando há cinco JournalNodes, a falha de gravação de dados de dois JournalNodes é permitida.

  O JournalNode é um processo de daemon leve e compartilha um host com outros serviços do Hadoop. Recomenda-se que o JournalNode seja implementado no nó de controle para evitar falhas na gravação de dados no JournalNode.

  Figura 6-38 Arquitetura do HDFS baseada em QJM



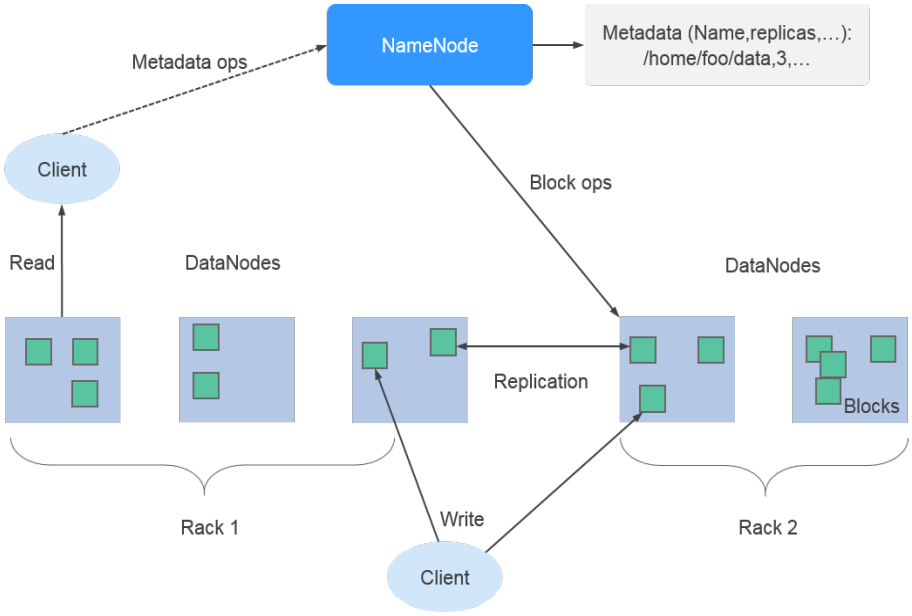
Princípio

O MRS usa o mecanismo de cópia do HDFS para garantir a confiabilidade dos dados. Um arquivo de backup é gerado automaticamente para cada arquivo salvo no HDFS, ou seja, duas

cópias são geradas no total. O número de cópias do HDFS pode ser consultado usando o parâmetro **dfs.replication**.

- Quando a especificação de nó central do cluster de MRS está definida como unidade de disco rígido não local (HDD) e o cluster tem apenas um nó central, o número padrão de cópias de HDFS é 1. Se o número de nós centrais no cluster for maior ou igual a 2, o número padrão de cópias de HDFS será 2.
- Quando a especificação do nó central do cluster do MRS é definida como disco local e o cluster tem apenas um nó central, o número padrão de cópias de HDFS é 1. Se houver dois nós centrais no cluster, o número padrão de cópias do HDFS é 2. Se o número de nós centrais no cluster for maior ou igual a 3, o número padrão de cópias HDFS será 3.

Figura 6-39 Arquitetura do HDFS



O componente do HDFS do MRS suporta os seguintes recursos:

- Suporta código de eliminação, reduzindo a redundância de dados para 50% e melhorando a confiabilidade. Além disso, a estrutura de armazenamento de blocos distribuídos é introduzida para maximizar o uso da capacidade de um único nó e vários discos em um cluster existente. Depois que o processo de codificação é introduzido, o desempenho de gravação de dados é melhorado e o desempenho é próximo ao da redundância de multi-cópias.
- Suporta programação balanceada de nó em HDFS e programação balanceada de disco em um único nó, melhorando o desempenho do armazenamento do HDFS após escalabilidade de nó ou disco.

Para obter detalhes sobre a arquitetura e os princípios do Hadoop, consulte **https://hadoop.apache.org/**.

6.9.2 Solução HA do HDFS

Plano de fundo de HA do HDFS

Em versões anteriores ao Hadoop 2.0.0, o SPOF ocorre no cluster HDFS. Cada cluster tem apenas um NameNode. Se o host onde o NameNode está localizado estiver com defeito, o

cluster do HDFS não poderá ser usado a menos que o NameNode seja reiniciado ou iniciado em outro host. Isso afeta a disponibilidade geral do HDFS nos seguintes aspectos:

1. No caso de um evento não planejado, como detalhamento do host, o cluster ficará indisponível até que o NameNode seja reiniciado.
2. Tarefas de manutenção planejadas, como atualização de software e hardware, farão com que o cluster pare de funcionar.

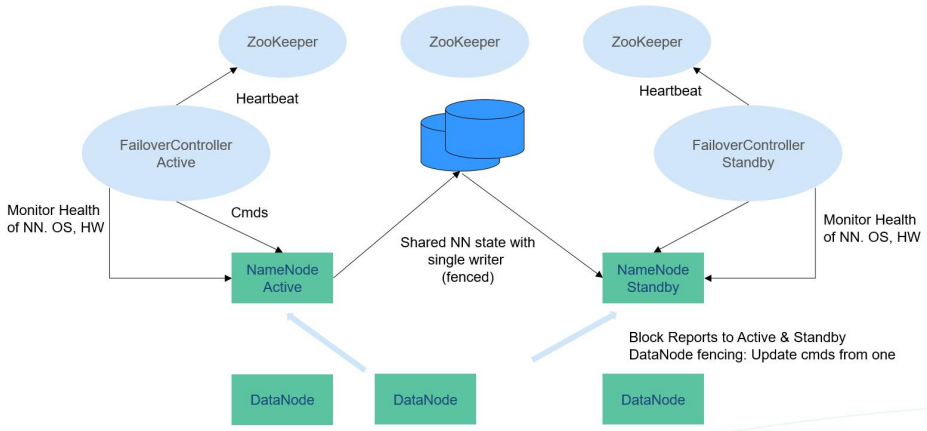
Para resolver os problemas anteriores, a solução HA do HDFS permite um backup de intercambiar do NameNode para NameNodes em um cluster em modo automático ou manual (configurável). Quando uma máquina falha (devido a uma falha de hardware), o NameNode ativo/em espera muda automaticamente em um curto espaço de tempo. Quando o NameNode ativo precisa ser mantido, o administrador do cluster do MRS pode executar manualmente uma alternância de NameNode ativo/em espera para garantir a disponibilidade do cluster durante a manutenção.

Para obter detalhes sobre failover automático do HDFS, consulte

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

Implementação de HA do HDFS

Figura 6-40 Implementação típica de HA



Em um cluster HA típico (como mostrado em **Figura 6-40**), dois NameNodes precisam ser configurados em dois servidores independentes, respectivamente. Em qualquer momento, um NameNode está no estado ativo e o outro NameNode está no estado de espera. O NameNode ativo é responsável por todas as operações do cliente no cluster, enquanto o NameNode em espera mantém a sincronização com o nó ativo para fornecer alternância rápida, se necessário.

Para manter os dados sincronizados uns com os outros, ambos os nós se comunicam com um grupo de JournalNodes. Quando o nó ativo modifica os metadados de qualquer sistema de arquivos, ele armazena o log de modificação na maioria desses JournalNodes. Por exemplo, se houver três JournalNodes, o log será salvo em dois deles pelo menos. O nó em espera monitora as alterações de JournalNodes e sincroniza as alterações do nó ativo. Com base no registro de modificação, o nó em espera aplica as alterações aos metadados do sistema de arquivos local. Uma vez que ocorre uma alternância, o nó em espera pode garantir que seu status seja o mesmo do nó ativo. Isso garante que os metadados do sistema de arquivos sejam sincronizados entre os nós ativo e em espera se a alternância for incorrida pela falha do nó ativo.

Para garantir uma alternância rápida, o nó em espera precisa ter as informações de bloco mais recentes. Portanto, os DataNodes enviam informações de bloqueio e mensagens de pulsação para dois NameNodes ao mesmo tempo.

É vital para um cluster HA que somente um dos NameNodes esteja ativo a qualquer momento. Caso contrário, o estado do namespace se dividiria em duas partes, arriscando a perda de dados ou outros resultados incorretos. Para evitar o chamado "cenário de dupla personalidade", o JournalNodes só permitirá que um único NameNode grave dados por vez. Durante a alternância, o NameNode que se tornará ativo assumirá o papel de gravar dados em JournalNodes. Isso efetivamente impede que o outro NameNodes esteja no estado ativo, permitindo que o novo nó ativo prossiga a alternância com segurança.

Para obter mais informações sobre a solução HA do HDFS, visite o seguinte site:

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

6.9.3 Relação entre HDFS e outros componentes

Relação entre HDFS e HBase

O HDFS é um subprojeto do Apache Hadoop, que é usado como o sistema de armazenamento de arquivos para o HBase. O HBase está localizado na camada de armazenamento estruturado. O HDFS fornece suporte altamente confiável para armazenamento de camada inferior do HBase. Todos os arquivos de dados do HBase podem ser armazenados no HDFS, exceto alguns arquivos de log gerados pelo HBase.

Relação entre HDFS e MapReduce

- O HDFS possui alta tolerância a falhas e alta taxa de transferência e pode ser implementado em hardware de baixo custo para armazenar dados de aplicações com conjuntos de dados massivos.
- MapReduce é um modelo de programação usado para computação paralela de grandes conjuntos de dados (maiores que 1 TB). Os dados computados pelo MapReduce vêm de várias fontes de dados, como FileSystem locais, HDFS e bancos de dados. A maioria dos dados vem do HDFS. A alta taxa de transferência do HDFS pode ser usada para ler dados massivos. Depois de computados, os dados podem ser armazenados no HDFS.

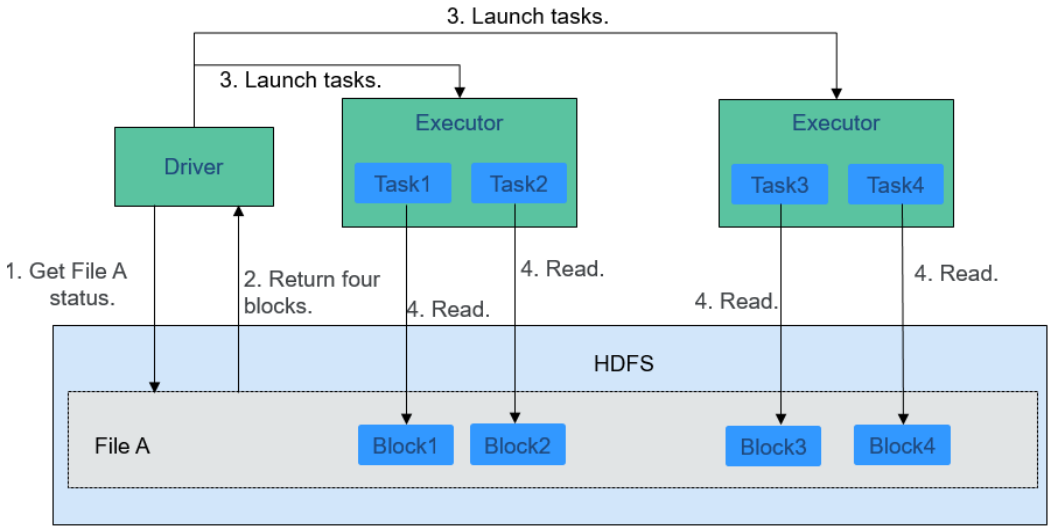
Relação entre HDFS e Spark

Os dados calculados pelo Spark vêm de várias fontes de dados, como arquivos locais e HDFS. A maioria dos dados vem do HDFS, que pode ler dados em larga escala para computação paralela. Depois de computados, os dados podem ser armazenados no HDFS.

O Spark envolve o Driver e o Executor. O Driver agenda tarefas e o Executor executa tarefas.

Figura 6-41 mostra como os dados são lidos de um arquivo.

Figura 6-41 Processo de leitura de arquivos

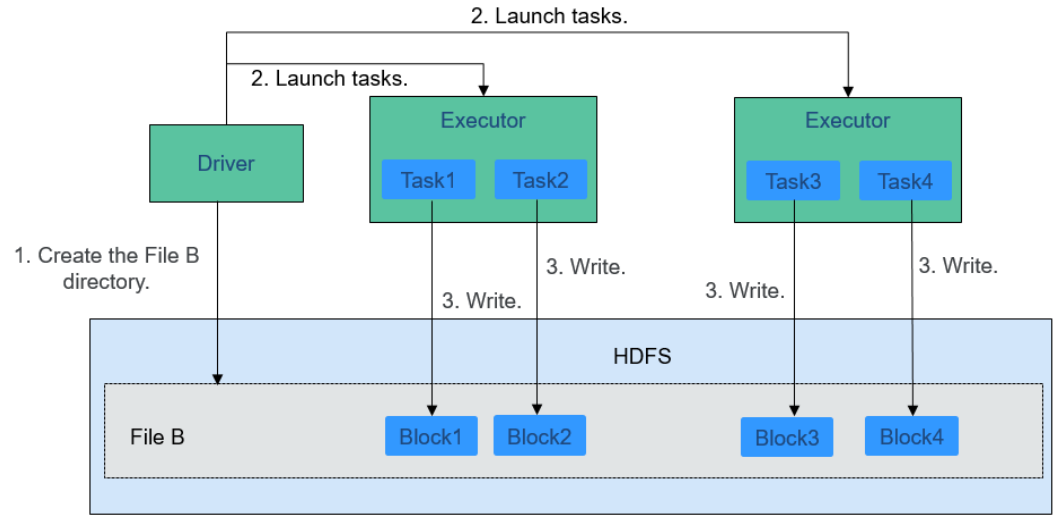


O processo de leitura do arquivo é o seguinte:

1. O Driver se interconecta com o HDFS para obter as informações do Arquivo A.
2. O HDFS retorna as informações de bloco detalhadas sobre esse arquivo.
3. O Driver define um grau paralelo com base na quantidade de dados do bloco e cria várias tarefas para ler os blocos desse arquivo.
4. O Executor executa as tarefas e lê os blocos detalhados como parte do Conjunto de dados distribuído resiliente (RDD).

Figura 6-42 mostra como os dados são gravados em um arquivo.

Figura 6-42 Processo de gravação de arquivos



O processo de escrita do arquivo é o seguinte:

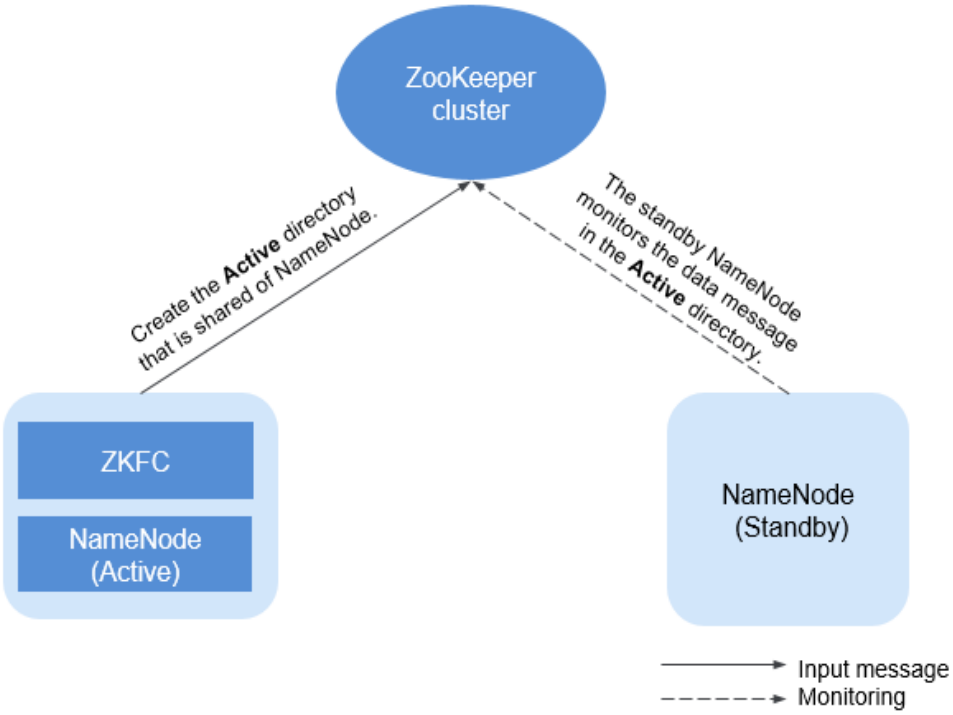
1. O Driver cria um diretório onde o arquivo deve ser gravado.
2. Com base no status de distribuição do RDD, o número de tarefas relacionadas à gravação de dados é calculado e essas tarefas são enviadas ao Executor.

3. O Executor executa essas tarefas e grava os dados RDD calculados no diretório criado em 1.

Relação entre HDFS e ZooKeeper

Figura 6-43 mostra a relação entre o ZooKeeper e o HDFS.

Figura 6-43 Relação entre ZooKeeper e HDFS



Como cliente de um cluster de ZooKeeper, o ZKFC (ZKFailoverController) monitora o status do NameNode. O ZKFC é implementado apenas no nó em que reside o NameNode e nos NameNodes de HDFS ativos e em espera.

1. O ZKFC se conecta ao ZooKeeper e salva informações como nomes de host em ZooKeeper no diretório **/hadoop-ha**. O NameNode que cria o diretório primeiro é considerado como o nó ativo e o outro é o nó em espera. NameNodes lê as informações NameNode periodicamente através do ZooKeeper.
2. Quando o processo do nó ativo termina anormalmente, o NameNode em espera detecta alterações no diretório **/hadoop-ha** por meio do ZooKeeper e, em seguida, assume o serviço do NameNode ativo.

6.9.4 Recursos de código aberto aprimorados do HDFS

Recurso de código aberto aprimorado: colocação de bloco de ficheiros

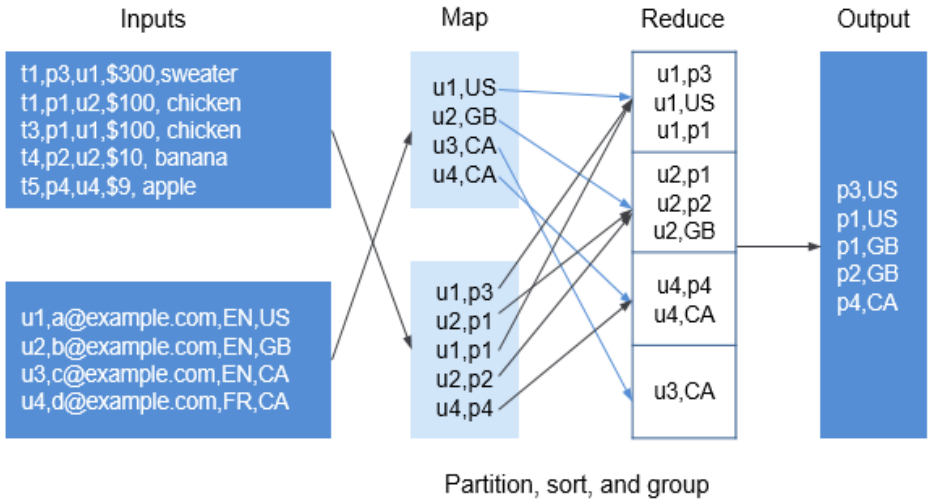
No cenário resumo de dados off-line e estatísticas, Join é uma função de computação usada com frequência e é implementada no MapReduce da seguinte maneira:

1. A tarefa Map processa os registros nos dois arquivos de tabela em Join Key e Value, executa o particionamento de hash por Join Key e envia os dados para diferentes tarefas Reduce para processamento.

2. As tarefas Reduce lêem dados na tabela à esquerda recursivamente no modo de loop aninhado e percorra cada linha da tabela à direita. Se os valores da chave de junção forem idênticos, os resultados da junção serão de saída.

   O método anterior reduz drasticamente o desempenho do cálculo da junção. Como uma grande quantidade de transferência de dados de rede é necessária quando os dados armazenados em nós diferentes são enviados do MAPA para Reduzir, como mostrado em **Figura 6-44**.

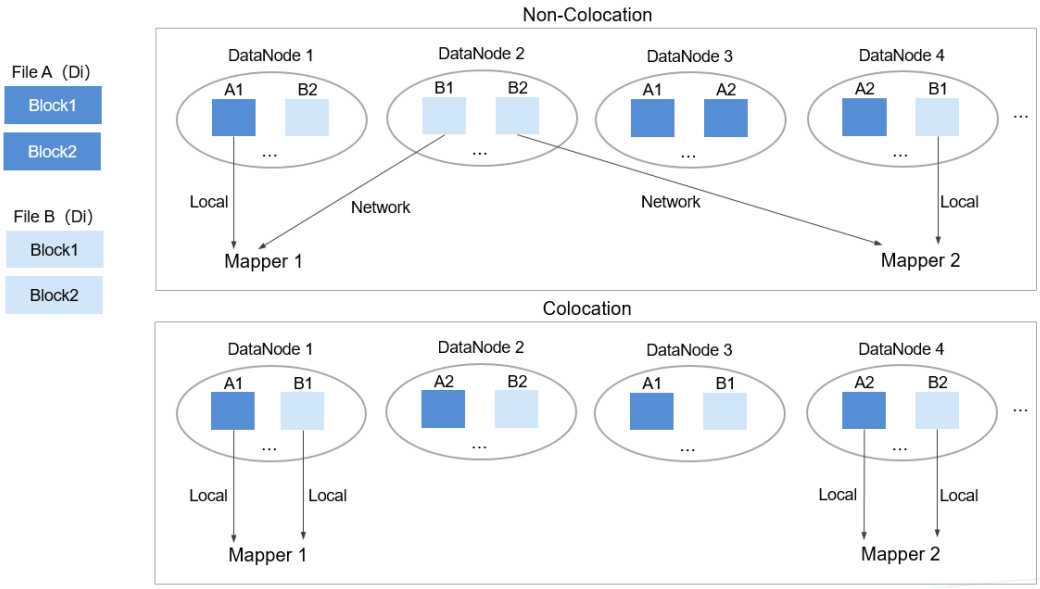
Figura 6-44 Transmissão de dados no cenário de não-colocação



As tabelas de dados são armazenadas no sistema de arquivos físico pelo bloco HDFS. Portanto, se dois blocos a serem juntados forem colocados no mesmo host de acordo depois que forem particionados pela chave de junção, você poderá obter os resultados diretamente da junção de Map no nó local sem qualquer transferência de dados no processo de redução do cálculo da junção. Isso vai melhorar muito o desempenho.

Com o mesmo recurso de distribuição dos dados do HDFS, um mesmo ID de distribuição é alocado para arquivos, FileA e FileB, nos quais os cálculos de associação e somatória precisam ser executados. Dessa forma, todos os blocos são distribuídos juntos e o cálculo pode ser feito sem recuperar dados entre nós, o que melhora muito o desempenho da junção de MapReduce.

Figura 6-45 Distribuição de blocos de dados em cenários de colocação e não-colocação

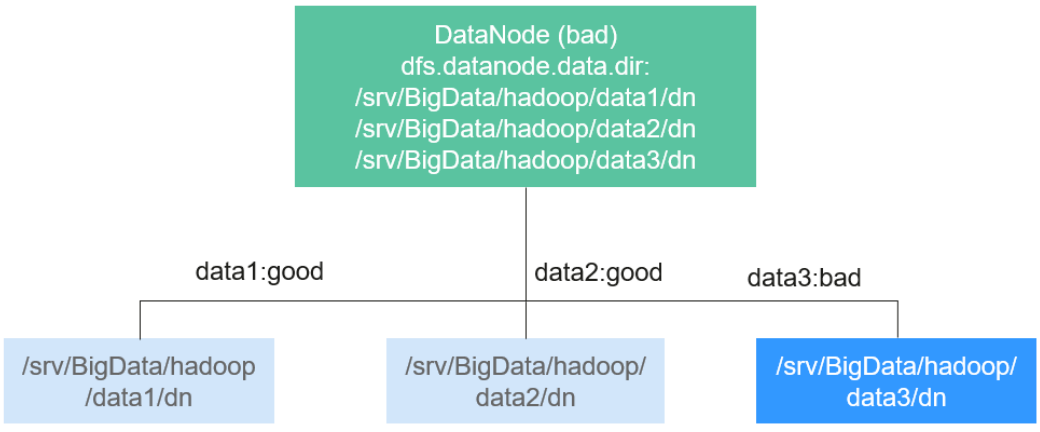


Recurso de código aberto aprimorado: configuração de volume de disco rígido danificado

Na versão de código aberto, se vários volumes de armazenamento de dados forem configurados para um DataNode o DataNode deixará de fornecer serviços por padrão se um dos volumes estiver danificado. Se o item de configuração **dfs.datanode.failed.volumes.tolerated** é definido para especificar o número de volumes danificados permitidos, o DataNode continua a fornecer serviços quando o número de volumes danificados não excede o limite.

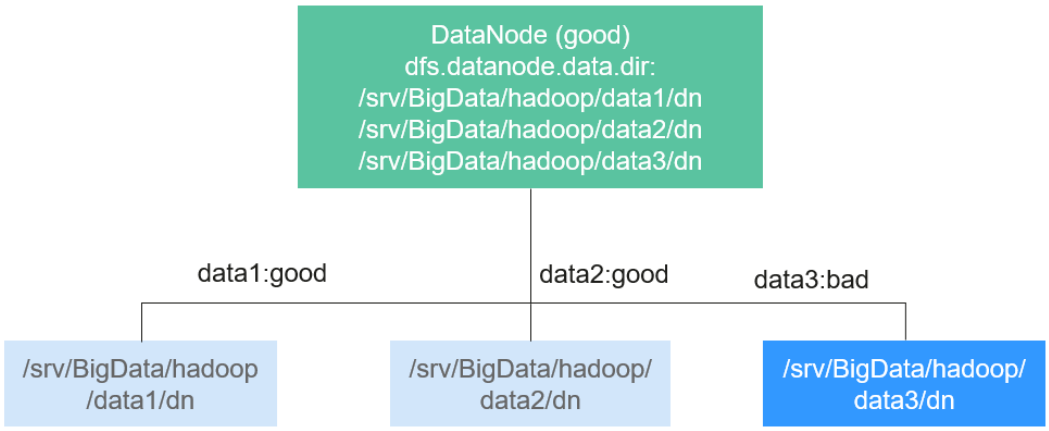
O valor de **dfs.datanode.failed.volumes.tolerated** varia de -1 ao número de volumes de disco configurados no DataNode. O valor padrão é **-1**, como mostrado em **Figura 6-46**.

Figura 6-46 Item sendo definido como 0



Por exemplo, três volumes de armazenamento de dados são montados em um DataNode e **dfs.datanode.failed.volumes.tolerated** é definido como 1. Nesse caso, se um volume de armazenamento de dados do DataNode não estiver disponível, esse DataNode ainda poderá fornecer serviços, conforme mostrado em **Figura 6-47**.

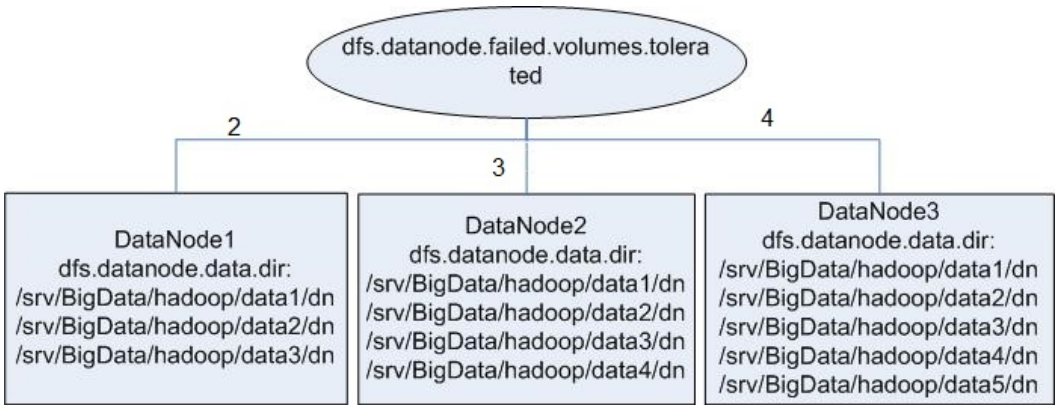
Figura 6-47 Item sendo definido como 1



Este item de configuração nativa tem alguns defeitos. Quando o número de volumes de armazenamento de dados em cada DataNode é inconsistente, é necessário configurar cada DataNode independentemente em vez de gerar o arquivo de configuração unificado para todos os nós.

Suponha que existem três DataNodes em um cluster. O primeiro nó tem três diretórios de dados, o segundo nó tem quatro e o terceiro nó tem cinco. Se quiser garantir que os serviços de DataNode estejam disponíveis quando apenas um diretório de dados estiver disponível, você precisa executar a configuração conforme mostrado em **Figura 6-48**.

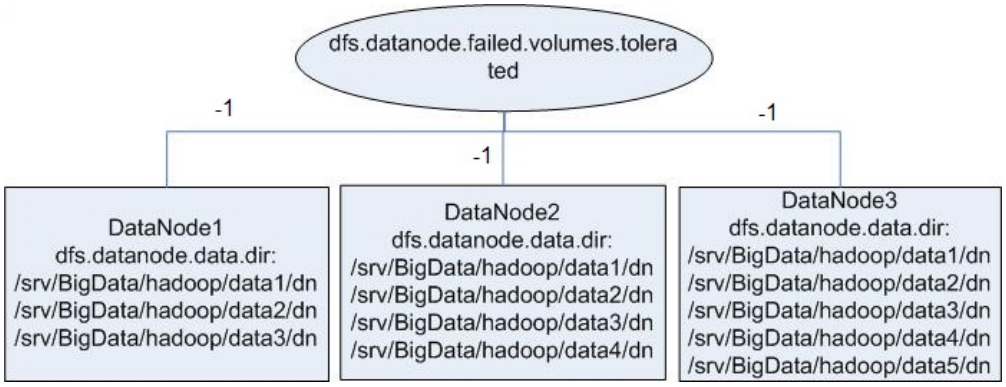
Figura 6-48 Configuração de atributos antes de ser aprimorada



No HDFS aprimorado auto-desenvolvido, esse item de configuração é aprimorado, com um valor **-1** adicionado. Quando este item de configuração é definido como **-1**, todos os DataNodes podem fornecer serviços, desde que um volume de armazenamento de dados em todos os DataNodes esteja disponível.

Para resolver o problema no exemplo anterior, defina essa configuração como **-1**, conforme mostrado em **Figura 6-49**.

Figura 6-49 Configuração de atributo após ser aprimorada



Recurso de código aberto aprimorado: aceleração de inicialização do HDFS

No HDFS, quando o NameNodes é iniciado, o arquivo de metadados FsImage precisa ser carregado. Em seguida, o DataNodes reportará as informações do bloco de dados após a inicialização do DataNodes. Quando as informações do bloco de dados reportadas pelo DataNodes atingem a porcentagem predefinida, o NameNodes sai do modo de segurança para concluir o processo de inicialização. Se o número de arquivos armazenados no HDFS atingir o nível de milhões ou bilhões, os dois processos são demorados e levarão a um longo tempo de inicialização do NameNode. Portanto, esta versão otimiza o processo de carregamento de FsImage de arquivos de metadados.

No HDFS de código aberto, o FsImage armazena todos os tipos de informações de metadados. Cada tipo de informação de metadados (como informações de metadados de arquivo e informações de metadados de pasta) é armazenada em um bloco de seção, respectivamente. Esses blocos de seção são carregados no modo serial durante a inicialização. Se um grande número de arquivos e pastas estiverem armazenados no HDFS, o carregamento das duas seções é demorado, prolongando o tempo de inicialização do HDFS. O NameNode do HDFS divide cada tipo de metadados por segmentos e armazena os dados em várias seções ao gerar os arquivos FsImage. Quando o NameNodes é iniciado, as seções são carregadas no modo paralelo. Isso acelera a inicialização do HDFS.

Recurso de código aberto aprimorado: políticas de colocação de blocos baseadas em rótulo (HDFS Nodelabel)

Você precisa configurar os nós para armazenar blocos de dados de arquivos do HDFS com base em recursos de dados. É possível configurar uma expressão de rótulo para um diretório ou arquivo do HDFS e atribuir um ou mais rótulos a um DataNode para que os blocos de dados do arquivo sejam armazenados em DataNodes especificados. Se a política de posicionamento de bloco de dados baseada em rótulo for usada para selecionar DataNodes para armazenar os arquivos especificados, o intervalo do DataNode será especificado com base na expressão de rótulo. Em seguida, os nós adequados são selecionados a partir do intervalo especificado.

- Você pode armazenar as réplicas de blocos de dados nos nós com rótulos diferentes de acordo. Por exemplo, armazene duas réplicas do bloco de dados para o nó rotulado com L1 e armazene outras réplicas do bloco de dados para os nós rotulados com L2.
- Você pode definir a política em caso de falha de posicionamento de bloco, por exemplo, selecione um nó de todos os nós aleatoriamente.

Figura 6-50 dá um exemplo:

- Os dados em **/HBase** são armazenados em A, B e D.
- Os dados em **/Spark** são armazenados em A, B, D, E e F.
- Os dados em **/user** são armazenados em C, D e F.
- Os dados em **/user/shl** são armazenados em A, E e F.

Figura 6-50 Exemplo de política de posicionamento de bloco baseada em rótulo

Recurso de código aberto aprimorado: HDFS Load Balance

As políticas atuais de leitura e gravação do HDFS são principalmente para otimização local sem considerar a carga real de nós ou discos. Com base nas cargas de I/O de diferentes nós, o balanceamento de carga do HDFS garante que, quando as operações de leitura e gravação forem executadas no cliente HDFS, o nó com baixa carga de I/O é selecionado para executar tais operações para equilibrar a carga de I/O e utilizar totalmente a taxa de transferência geral do cluster.

Se o HDFS Load Balance estiver ativado durante a gravação do arquivo, o NameNode selecionará um DataNode (na ordem de nó local, rack local e rack remoto). Se a carga de I/O do nó selecionado for pesada, o NameNode escolherá outro DataNode com carga mais leve.

Se o HDFS Load Balance estiver ativado durante a leitura do arquivo, um cliente de HDFS enviará uma solicitação ao NameNode para fornecer a lista de DataNodes que armazenam o bloco a ser lido. O NameNode retorna uma lista de DataNodes ordenados por distância na topologia de rede. Com o recurso de HDFS Load Balance, os DataNodes na lista também são classificados por sua carga de I/O. Os DataNodes com carga pesada estão na parte inferior da lista.

Caraterísticas de código aberto aprimoradas: movimentação automática de dados do HDFS

O Hadoop tem sido usado para processamento em lote de imensos dados há muito tempo. O modelo do HDFS existente é usado para atender muito bem às necessidades das aplicações de processamento em lote, porque essas aplicações se concentram mais na taxa de transferência do que no atraso.

No entanto, como o Hadoop é cada vez mais usado para aplicações de camada superior que exigem acesso aleatório frequente de I/O, como Hive e HBase, discos de baixa latência, como discos de estado sólido (SSD), são favorecidos em cenários sensíveis a atrasos. Para atender à tendência, o HDFS suporta uma variedade de tipos de armazenamento. Os usuários podem escolher um tipo de armazenamento de acordo com suas necessidades.

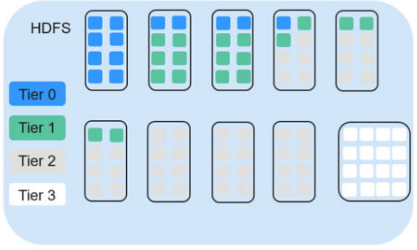
As políticas de armazenamento variam dependendo da frequência com que os dados são usados. Por exemplo, se os dados acessados com frequência no HDFS forem marcados como **ALL_SSD** ou **HOT**, os dados acessados várias vezes poderão ser marcados como **WARM** e os dados raramente acessados (apenas acesso uma ou duas vezes) poderão ser marcados como **COLD**. Você pode selecionar diferentes políticas de armazenamento de dados com base na frequência de acesso aos dados.

No entanto, os discos de baixa latência são muito mais caros do que os discos giratórios. Os dados geralmente apresentam um uso inicial intenso com declínio no uso ao longo de um período de tempo. Portanto, pode ser útil se os dados que não são mais usados forem movidos de discos caros para mídias de armazenamento mais baratas.

Um exemplo típico é o armazenamento de registros detalhados. Novos registros de detalhes são importados para o SSD porque são frequentemente consultados por aplicações de camada superior. À medida que a frequência de acesso a esses registros detalhados diminui, eles são movidos para um armazenamento mais barato.

Antes que a movimentação automática de dados seja alcançada, você precisa determinar manualmente por tipo de serviço se os dados são usados com frequência, definir manualmente uma política de armazenamento de dados e acionar manualmente a ferramenta HDFS Auto Data Movement, conforme mostrado na figura abaixo.

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>



Se os dados envelhecidos puderem ser identificados automaticamente e transferidos para armazenamento mais barato (como disco/arquivamento), você verá cortes significativos de custos e melhoria da eficiência do gerenciamento de dados.

A ferramenta HDFS Auto Data Movement está no centro da movimentação automática de dados do HDFS. Ele define automaticamente uma política de armazenamento dependendo da frequência com que os dados são usados. Especificamente, as funções da ferramenta HDFS Auto Data Movement podem:

- Marcar uma política de armazenamento de dados como **All_SSD**, **One_SSD**, **Hot**, **Warm**, **Cold** ou **FROZEN** de acordo com a idade, o tempo de acesso e as regras de movimentação manual de dados.
- Definir regras para distinguir dados frios e quentes com base na idade dos dados, no tempo de acesso e nas regras de migração manual.
- Definir a ação a ser tomada se as regras baseadas na idade forem atendidas.

  MARK: a ação para identificar se os dados são usados com frequência ou raramente com base nas regras de idade e definir uma política de armazenamento de dados.
  MOVE: a ação para invocar a ferramenta HDFS Auto Data Movement e mover dados com base nas regras de idade para identificar se os dados são usados com frequência ou raramente depois de ter determinado a política de armazenamento correspondente.
  - **MARK**: identifica se os dados são usados com frequência ou raramente e define a política de armazenamento de dados.
  - **MOVE**: a ação para invocar a Ferramenta de Movimento Automático de Dados do HDFS e mover dados entre camadas.
  - **SET_REPL**: a ação para definir uma nova quantidade de réplicas para um arquivo.
  - **MOVE_TO_FOLDER**: a ação para mover arquivos para uma pasta de destino.
  - **DELETE**: a ação para excluir um arquivo ou diretório.
  - **SET_NODE_LABEL**: a ação para definir rótulos de nó de um arquivo.

Com o recurso de movimentação automática de dados do HDFS, você só precisa definir a idade com base nas regras de tempo de acesso. A ferramenta HDFS Auto Data Movement combina dados de acordo com regras baseadas em idade, define políticas de armazenamento e move dados. Dessa forma, a eficiência do gerenciamento de dados e a eficiência dos recursos do cluster são aprimoradas.

6.10 HetuEngine

6.10.1 Visão geral do produto HetuEngine

Esta secção aplica-se apenas à MRS 3.1.2-LTS.3.

Descrição de HetuEngine

HetuEngine é um mecanismo de virtualização de dados e análise de SQL interativa de alto desempenho interno. Ele se integra perfeitamente ao ecossistema de Big Data para implementar consultas interativas de grandes quantidades de dados em segundos e suporta acesso unificado a dados entre fontes e entre domínios para permitir uma análise única de convergência de SQL no data lake, entre lakes e entre lakehouses.

Arquitetura de HetuEngine

HetuEngine consiste em diferentes módulos. **Figura 6-51** mostra a arquitetura.

Figura 6-51 Arquitetura de HetuEngine

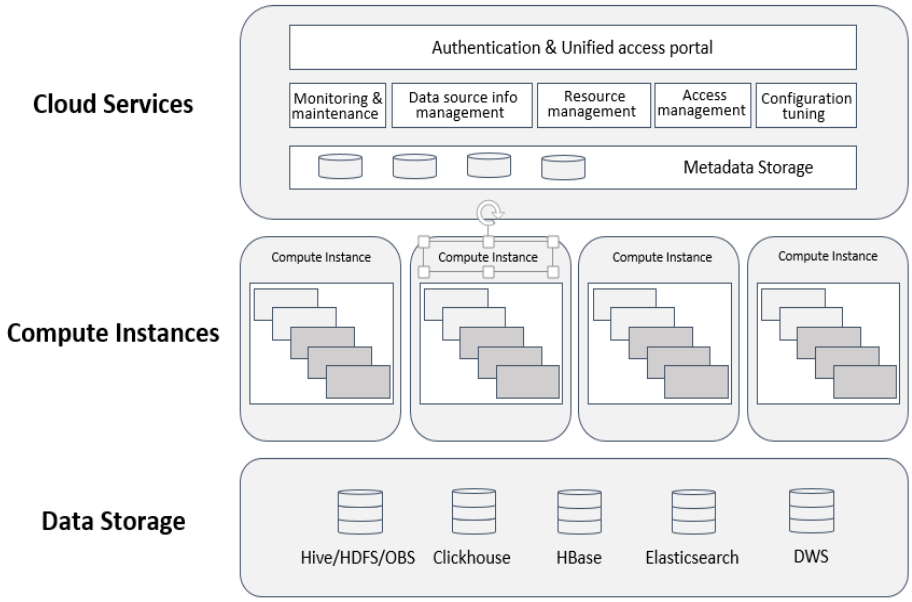


Tabela 6-10 Descrição de módulo

Módulo	Conceito	Descrição
Camada de serviço de nuvem	HetuEngine CLI/ JDBC	Cliente de HetuEngine, através do qual a solicitação de consulta é enviada e os resultados são retornados e exibidos.
	HSBroker	Componente de gerenciamento de serviços de HetuEngine. Ele gerencia e verifica instâncias de computação, monitora o status de integridade e realiza manutenção automática.
	HSConsole	Fornece GUIs de operação visualizadas e APIs RESTful para gerenciamento de informações de fontes de dados, gerenciamento de instâncias de computação e consulta automática de tarefas.
	HSFabric	Fornece de alto desempenho entre domínios (centros de dados).
Camada do motor	Coordinator	Nó de gerenciamento de instâncias de computação de HetuEngine. Ele recebe e analisa instruções de SQL, gera e otimiza planos de execução, atribui tarefas e agenda recursos.
	Worker	Nó de trabalho das instâncias de computação de HetuEngine. Ele fornece recursos como extração de dados paralelos de fontes de dados e computação de SQL distribuída.

Cenários de aplicação de HetuEngine

HetuEngine suporta consulta conjunta rápida entre fontes (várias fontes de dados, como Hive, HBase, GaussDB (DWS), Elasticsearch e ClickHouse) e entre domínios (várias regiões ou data centers), especialmente para consulta rápida interativa de dados do Hive e Hudi no cluster de Hadoop (HetuEngine).

6.10.2 Relação entre HetuEngine e outros componentes

A instalação de HetuEngine depende do cluster do MRS. **Tabela 6-11** lista os componentes dos quais depende a instalação do HetuServer.

Tabela 6-11 Componentes dos quais HetuEngine depende

Nome	Descrição
HDFS	Hadoop Distributed File System, suportando acesso a dados de alta taxa de transferência e adequado para aplicações com conjuntos de dados de grande escala.
Hive	Data warehouse de código aberto construído em Hadoop. Ele armazena dados estruturados e implementa análise básica de dados usando a Hive Query Language (HQL), uma linguagem semelhante a SQL.
ZooKeeper	Permite coordenação distribuída altamente confiável. Ajuda a evitar pontos únicos de falhas (SPOFs) e fornece serviços confiáveis para aplicações.
KrbServer	Centro de gerenciamento de chaves que distribui contas.
Yarn	Sistema de gerenciamento de recursos, que é um módulo de recursos geral que gerencia e agenda recursos para várias aplicações.
DBService	O DBService é um sistema de armazenamento de banco de dados relacional de alta disponibilidade que fornece funções de backup e restauração de metadados.

6.11 Hive

6.11.1 Princípios básicos do Hive

O Hive é uma infraestrutura de armazém de dados construída sobre o Hadoop. Ele fornece uma série de ferramentas que podem ser usadas para extrair, transformar e carregar dados (ETL). Hive é um mecanismo que pode armazenar, consultar e analisar dados em massa armazenados no Hadoop. Hive define linguagem de consulta simples como SQL, que é conhecido como HiveQL. Ele permite que um usuário familiarizado com SQL consulte dados. A computação de dados do Hive depende do MapReduce, Spark e Tez.

O novo motor de execução Tez é usado para substituir o MapReduce original, melhorando significativamente o desempenho. O Tez pode converter vários jobs dependentes em um job,

portanto, apenas quando a gravação do HDFS for necessária e menos nós de trânsito forem necessários, melhorando significativamente o desempenho dos jobs de DAG.

Hive fornece as seguintes funções:

- Analisa dados estruturados massivos e resume os resultados da análise.
- Permite que jobs complexas do MapReduce sejam compilados em linguagens SQL.
- Suporta formatos flexíveis de armazenamento de dados, incluindo notação de objeto JavaScript (JSON), valores separados por vírgulas (CSV), TextFile, RCFile, SequenceFile e ORC (Optimized Row Columnar).

Estrutura do sistema do Hive:

- Interface de usuário: três interfaces de usuário estão disponíveis, ou seja, CLI, Cliente e WUI. A CLI é a interface de usuário mais usada. Uma transcrição do Hive é iniciada quando a CLI é iniciada. Cliente refere-se a um cliente Hive, e um usuário cliente se conecta ao Servidor Hive. Ao entrar no modo cliente, você precisa especificar o nó em que o Servidor Hive reside e iniciar o Servidor Hive nesse nó. A IU da Web é usada para acessar o Hive através de um navegador. O MRS pode acessar o Hive somente no modo de cliente. Para obter detalhes, consulte **Uso do Hive a partir rascunho**. Para obter detalhes sobre como desenvolver aplicação do Hive, consulte **Desenvolvimento de aplicação do Hive**.
- Armazenamento de metadados: o Hive armazena metadados em bancos de dados, por exemplo, MySQL e Derby. Metadados no Hive incluem um nome de tabela, colunas de tabela e partições e suas propriedades, propriedades de tabela (indicando se uma tabela é uma tabela externa) e o diretório onde os dados da tabela são armazenados.

Estrutura do Hive

O Hive é um processo de serviço de instância única que fornece serviços traduzindo o HQL em jobs de MapReduce relacionados ou operações de HDFS. **Figura 6-52** mostra como o Hive está conectado a outros componentes.

Figura 6-52 Estrutura do Hive

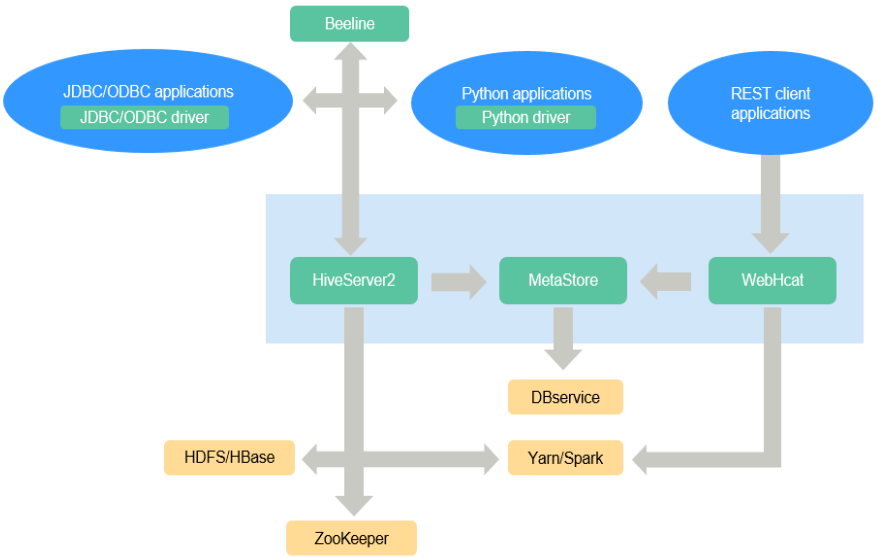
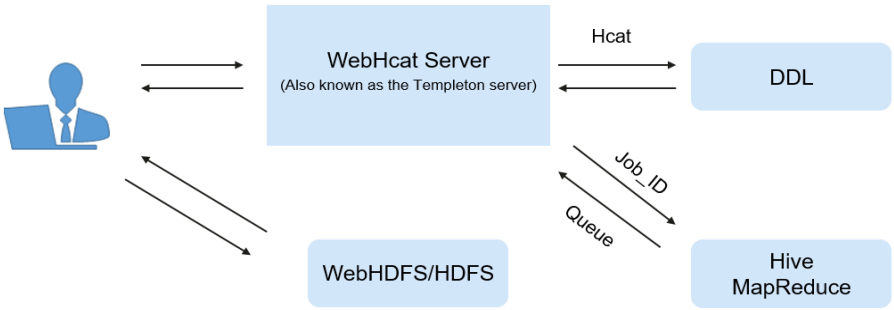


Tabela 6-12 Descrição do módulo

Módulo	Descrição
HiveServer	Vários HiveServers podem ser implantados em um cluster para compartilhar cargas. O HiveServer fornece serviços de banco de dados do Hive externamente, converte instruções HQL em tarefas do YARN ou operações do HDFS para concluir a extração, conversão e análise de dados.
MetaStore	● Vários MetaStores podem ser implantados em um cluster para compartilhar cargas. O MetaStore fornece serviços de metadados do Hive, além de ler, gravar, manter e modificar a estrutura e as propriedades das tabelas do Hive. ● MetaStore fornece APIs Thrift para HiveServer, Spark, WebHCat e outros clientes de MetaStore para acessar e operar metadados.
WebHCat	Vários WebHCats podem ser implantados em um cluster para compartilhar cargas. O WebHCat fornece APIs REST e executa os comandos do Hive por meio das APIs REST para enviar jobs do MapReduce.
Cliente do Hive	O cliente do Hive inclui a interface de linha de comando (CLI) homem-máquina Beeline, unidade JDBC para aplicações JDBC, driver Python para aplicações Python e arquivos JAR HCatalog para MapReduce.
Cluster do ZooKeeper	Como um nó temporário, o ZooKeeper registra a lista de endereços IP de cada instância do HiveServer. O driver do cliente se conecta a ZooKeeper para obter a lista e seleciona instâncias de HiveServer correspondentes com base no mecanismo de roteamento.
Cluster do HDFS/HBase	O cluster do HDFS armazena os dados da tabela do Hive.
Cluster do MapReduce/ YARN	Fornece serviços de computação distribuída. A maioria das operações de dados do Hive depende do MapReduce. A principal função do HiveServer é traduzir instruções HQL em jobs do MapReduce para processar dados massivos.

HCatalog é construído no Hive Metastore e incorpora a capacidade DDL do Hive. HCatalog também é uma tabela baseada em Hadoop e camada de gerenciamento de armazenamento que permite leitura/gravação de dados conveniente em tabelas de HDFS usando diferentes ferramentas de processamento de dados, como MapReduce e Pig. Além disso, o HCatalog também fornece APIs de leitura/gravação para essas ferramentas e usa uma CLI do Hive para publicar comandos para definir dados e consultar metadados. Depois de encapsular esses comandos, o WebHCat Server pode fornecer APIs RESTful, conforme mostrado em **Figura 6-53**.

Figura 6-53 Arquitetura lógica de WebHCat



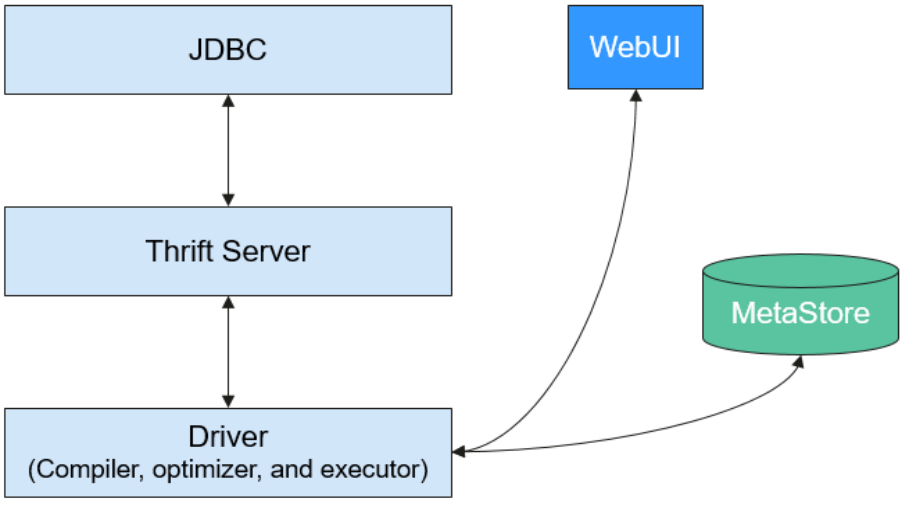
Princípios

O Hive funciona como um armazém de dados baseado na arquitetura HDFS e MapReduce e converte instruções HQL em jobs do MapReduce ou operações HDFS. Para obter detalhes sobre o Hive e o HQL, consulte **Manual de linguagem HiveQL**.

Figura 6-54 mostra a estrutura do Hive.

- **Metastore**: lê, grava e atualiza metadados como tabelas, colunas e partições. Sua camada inferior são os bancos de dados relacionais.
- **Driver**: gerencia o ciclo de vida da execução da HiveQL e participa de toda a execução do job do Hive.
- **Compiler**: traduz instruções HQL em uma série de jobs interdependentes de Map ou Reduce.
- **Optimizer**: é classificado em otimizador lógico e otimizador físico para otimizar planos de execução HQL e jobs de MapReduce, respectivamente.
- **Executor**: executa jobs Map ou Reduce com base em dependências de job.
- **ThriftServer** funciona como os servidores de JDBC, fornece APIs Thrift e integra-se com Hive e outras aplicações.
- **Clients**: inclui as APIs da WebUI e JDBC e fornece APIs para acesso do usuário.

Figura 6-54 Estrutura do Hive



6.11.2 Princípios do CBO do Hive

Princípios do CBO do Hive

CBO é a abreviação de Cost-Based Optimization.

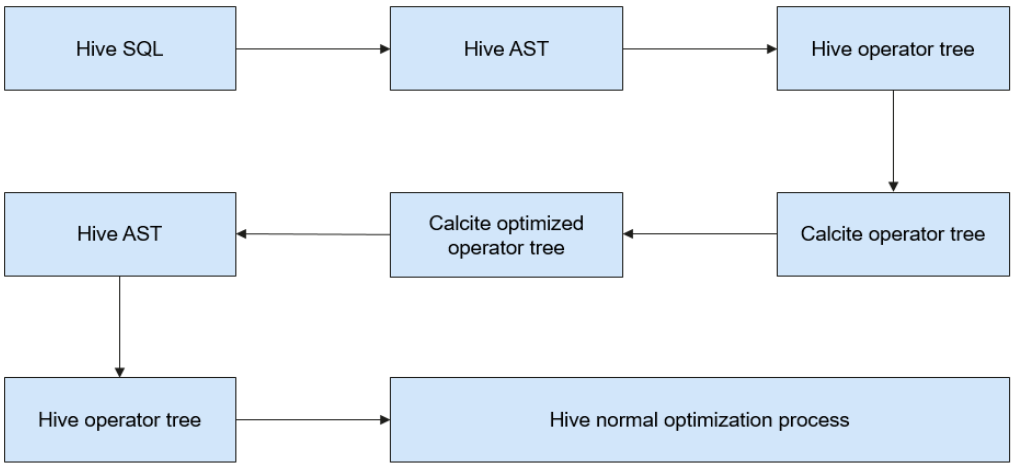
Isso otimizará o seguinte:

durante a compilação, o CBO calcula a sequência de junção mais eficiente com base em tabelas e condições de consulta envolvidas em instruções de consulta para reduzir o tempo e os recursos necessários para a consulta.

No Hive, o CBO é implementado da seguinte forma:

o Hive usa o componente de código aberto Apache Calcite para implementar o CBO. As instruções SQL são primeiro convertidas em Árvores de Sintaxe Abstratas do Hive (ASTs) e depois em RelNodes que podem ser identificadas por Calcite. Depois que o RelNodes ajusta a sequência de junção, os RelNodes são convertidos em ASTs pelo Hive para continuar a otimização lógica e física. **Figura 6-55** mostra o fluxo de trabalho.

Figura 6-55 Processo de implementação do CBO



Calcite ajusta a sequência de junção da seguinte forma:

1. Uma tabela é selecionada como a primeira tabela das tabelas a serem unidas.
2. A segunda e terceira tabelas são selecionadas com base no custo. Desta forma, vários planos de execução diferentes são obtidos.
3. Um plano com os custos mínimos é calculado e serve como a sequência final.

O método de cálculo de custos é o seguinte:

na versão atual, os custos são medidos com base no número de entradas de dados após a adesão. Menos entradas de dados significam menos custo. O número de entradas de dados associadas depende da taxa de seleção de tabelas associadas. O número de entradas de dados em uma tabela é obtido com base nas estatísticas de nível de tabela.

O número de entradas de dados em uma tabela após a filtragem é estimado com base nas estatísticas de nível de coluna, incluindo os valores máximos (máx.), valores mínimos (min.) e Número de Valores Distintos (NDV).

Por exemplo, existe uma tabela **table_a** cujo número total de registros de dados é 1.000.000 e NDV é 50. As condições de consulta são as seguintes:

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

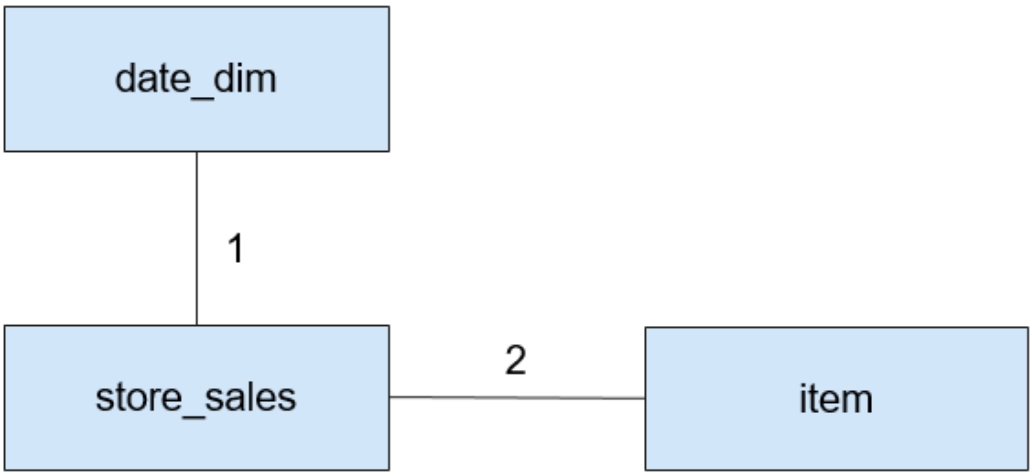
O número estimado de entradas de dados consultadas é: 1.000.000 × 1/50 = 20.000. A taxa de seleção é de 2%.

A seguir, o TPC-DS Q3 é um exemplo para descrever como o CBO ajusta a sequência de junção:

```
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;
```

Explicação da instrução: esta instrução indica que a junção interna é executada para três tabelas: tabela **store_sales** é uma tabela de fatos com cerca de 2.900.000.000 entradas de dados, tabela **date_dim** é uma tabela de dimensão com cerca de 73.000 entradas de dados, e a tabela **item** é uma tabela de dimensão com cerca de 18.000 entradas de dados. Cada tabela tem condições de filtragem. **Figura 6-56** mostra a relação de junção.

Figura 6-56 Relação da junção



O CBO deve primeiro selecionar as tabelas que trazem o melhor efeito de filtragem para a junção.

Analisando o min, máx, NDV e o número de entradas de dados, o CBO estima as taxas de seleção de diferentes tabelas de dimensão, como mostrado em **Tabela 6-13**.

Tabela 6-13 Filtragem de dados

Tabela	Número de entradas de dados originais	Número de entradas de dados após filtragem	Taxa de seleção
date_dim	73.000	6.200	8,5%
item	18.000	19	0,1%

A taxa de seleção pode ser estimada da seguinte forma: taxa de seleção = número de entradas de dados após a filtragem/número de entradas de dados originais

Conforme mostrado na tabela anterior, a tabela **item** tem um efeito de filtragem melhor. Portanto, o CBO ingressa na tabela **item** primeiro antes de ingressar na tabela **date_dim**.

Figura 6-57 mostra o processo de junção quando o CBO está desabilitado.

Figura 6-57 Processo de junção quando o CBO está desativado

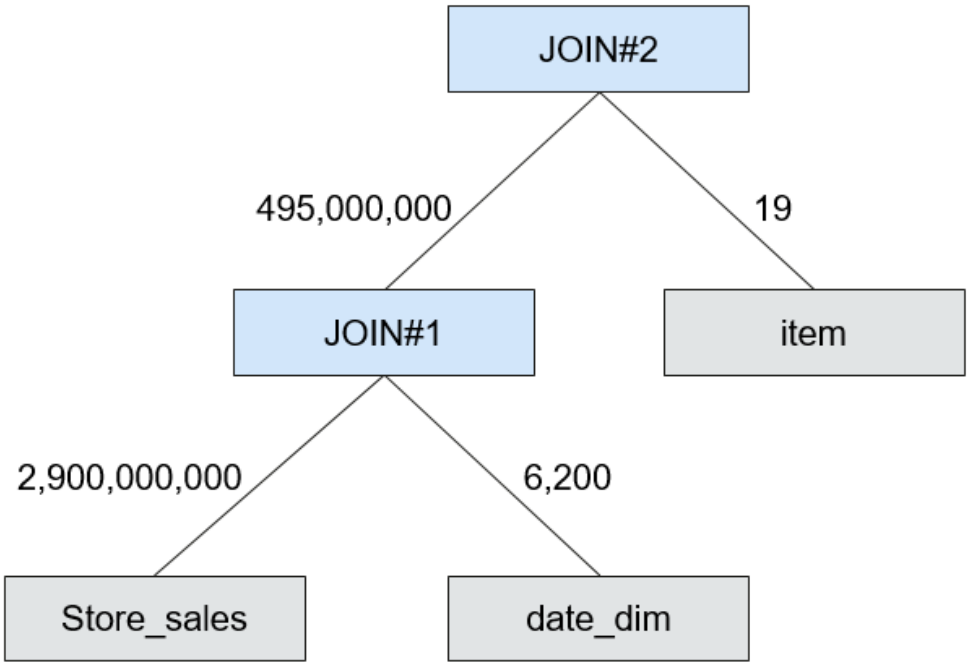
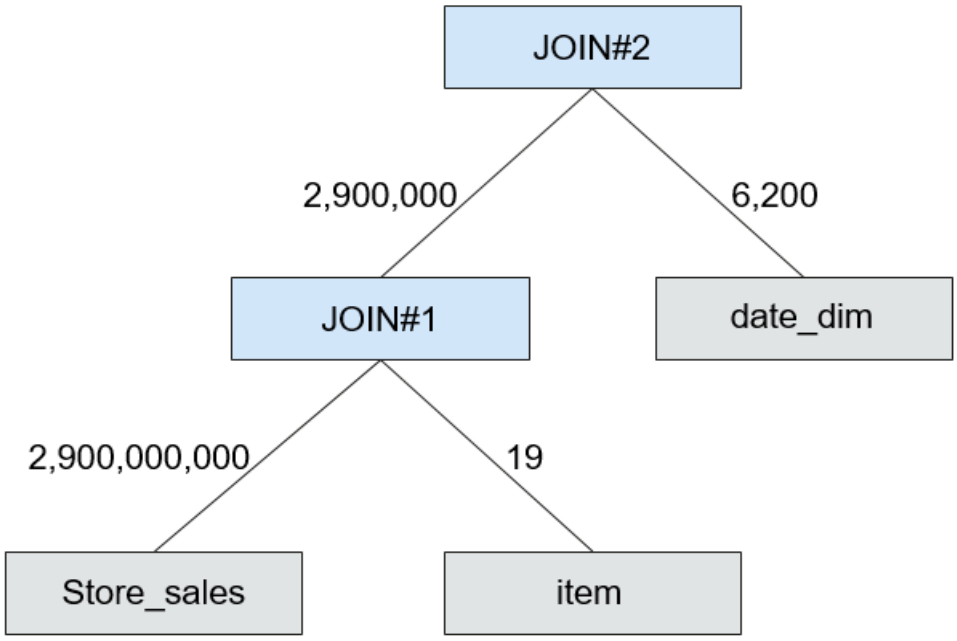


Figura 6-58 mostra o processo de junção quando o CBO está ativado.

Figura 6-58 Processo de junção quando o CBO está ativado



Depois que o CBO está ativado, o número de entradas de dados intermediários é reduzido de 495.000.000 para 2.900.000 e, assim, o tempo de execução pode ser notavelmente reduzido.

6.11.3 Relação entre Hive e outros componentes

Relação entre Hive e HDFS

O Hive é um subprojeto do Apache Hadoop, que usa o HDFS como sistema de armazenamento de arquivos. Ele analisa e processa dados estruturados com armazenamento subjacente altamente confiável suportado pelo HDFS. Todos os arquivos de dados no banco de dados do Hive são armazenados no HDFS, e todas as operações de dados no Hive também são realizadas usando APIs do HDFS.

Relação entre Hive e MapReduce

A computação de dados do Hive depende do MapReduce. MapReduce também é um sub-projeto do Apache Hadoop e é uma estrutura de computação paralela baseado em HDFS. Durante a análise de dados, o Hive analisa as instruções HQL enviadas pelos usuários em tarefas de MapReduce e envia as tarefas para o MapReduce executar.

Relação entre Hive e Tez

Tez, um projeto de código aberto do Apache, é uma estrutura de computação distribuída que suporta grafos acíclicos direcionados (DAGs). Quando o Hive usa o mecanismo Tez para analisar dados, ele analisa as instruções HQL enviadas pelos usuários em tarefas de Tez e envia as tarefas ao Tez para execução.

Relação entre Hive e DBService

O MetaStore (serviço de metadados) do Hive processa as informações de estrutura e atributo dos metadados do Hive, como bancos de dados, tabelas e partições do Hive. As informações precisam ser armazenadas em um banco de dados relacional e são gerenciadas e processadas pelo MetaStore. No produto, os metadados do Hive são armazenados e mantidos pelo componente DBService e o serviço de metadados é fornecido pelo componente Metadata.

6.11.4 Recurso de código aberto aprimorado

Recurso de código aberto aprimorado: colocação de HDFS

A colocação de HDFS é a função de controle de localização de dados fornecida pelo HDFS. A API de colocação de HDFS armazena dados associados ou dados nos quais as operações associadas são executadas no mesmo nó de armazenamento.

Hive suporta colocação de HDFS. Quando as tabelas do Hive são criadas, depois que as informações do localizador são definidas para arquivos de tabela, os arquivos de dados das tabelas relacionadas são armazenados no mesmo nó de armazenamento. Isso garante a computação de dados conveniente e eficiente entre as tabelas associadas.

Recurso de código aberto aprimorado: encriptação de colunas

O Hive oferece suporte à criptografia de uma ou mais colunas. As colunas a serem criptografadas e o algoritmo de criptografia podem ser especificados quando uma tabela de Hive é criada. Quando os dados são inseridos na tabela usando a instrução INSERT, as colunas relacionadas são criptografadas. A criptografia de coluna de Hive não suporta exibições e o cenário Hive sobre HBase.

O mecanismo de criptografia de coluna de Hive suporta dois algoritmos de criptografia que podem ser selecionados para atender aos requisitos do site durante a criação da tabela:

- AES (a classe de criptografia é **org.apache.hadoop.hive.serde2.AESRewriter**)
- SMS4 (a classe de criptografia é **org.apache.hadoop.hive.serde2.SMS4Rewriter**)

Recurso de código aberto aprimorado: exclusão de HBase

Devido às limitações dos sistemas de armazenamento subjacentes, o Hive não oferece suporte à capacidade de excluir uma única parte dos dados da tabela. No Hive do HBase, o Hive na solução MRS oferece suporte à capacidade de excluir uma única parte dos dados da tabela do HBase. Usando uma sintaxe específica, o Hive pode excluir um ou mais dados de uma tabela do HBase.

Recurso de código aberto aprimorado: delimitador de linha

Na maioria dos casos, um caractere de retorno de carro é usado como o delimitador de linha em tabelas de Hive armazenadas em arquivos de texto, ou seja, o caractere de retorno de carro é usado como o terminador de uma linha durante as consultas.

No entanto, alguns arquivos de dados são delimitados por caracteres especiais, e não por um caractere de retorno de carro.

O MRS Hive permite especificar diferentes caracteres ou combinações de caracteres como delimitadores de linha para dados do Hive em arquivos de texto.

Recurso de código aberto aprimorado: alternância de API REST baseada em HTTPS/HTTP

O WebHCat fornece APIs REST externas para o Hive. Por padrão, a versão da comunidade de código aberto usa o protocolo HTTP.

O MRS Hive suporta o protocolo HTTPS que é mais seguro e permite a alternância entre o protocolo HTTP e o protocolo HTTPS.

Recurso de código aberto aprimorado: função de transformar

A função de transformar não é permitida pelo Hive da versão de código aberto. O MRS Hive suporta a configuração da função de transformar. A função está desativada por padrão, que é a mesma da versão da comunidade de código aberto.

Os usuários podem modificar as configurações da função de transformar para habilitar a função. No entanto, existem riscos de segurança quando a função de transformar está ativada.

Recurso de código aberto aprimorado: criação de função temporária sem permissão ADMIN

Você deve ter permissão **ADMIN** ao criar funções temporárias no Hive da versão da comunidade de código aberto. O MRS Hive suporta a configuração da função para criar funções temporárias com permissão **ADMIN**. A função está desativada por padrão, que é a mesma da versão da comunidade de código aberto.

Você pode modificar as configurações desta função. Depois que a função estiver ativada, você poderá criar funções temporárias sem a permissão **ADMIN**.

Recurso de código aberto aprimorado: autorização do banco de dados

Na versão da comunidade de código aberto do Hive, somente o proprietário do banco de dados pode criar tabelas no banco de dados. Você pode receber as permissões **CREATE** e **SELECT** em tabelas pelo MRS Hive em um banco de dados. Depois que você recebe a permissão para consultar dados no banco de dados, o sistema associa automaticamente a permissão de consulta em todas as tabelas no banco de dados.

Recurso de código aberto aprimorado: autorização de coluna

A versão da comunidade de código aberto do Hive suporta apenas o controle de permissão no nível da tabela. O MRS Hive oferece suporte ao controle de permissão no nível da coluna. Você pode receber permissões de nível de coluna, como **SELECT**, **INSERT** e **UPDATE**.

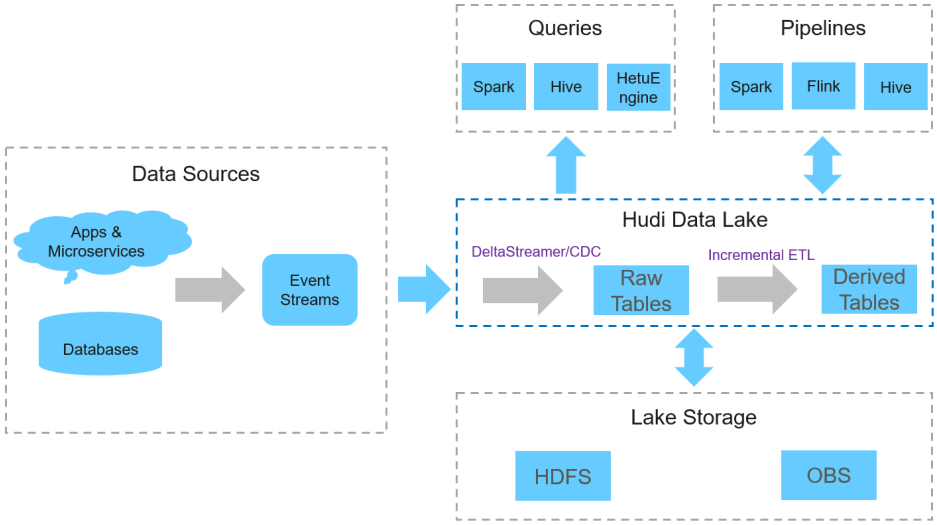
6.12 Hudi

Hudi é um formato de tabela de data lake que fornece a capacidade de atualizar e excluir dados, bem como consumir novos dados no HDFS. Ele suporta vários mecanismos de computação e fornece interfaces de inserção, atualização e exclusão (IUD) e primitivos de streaming, incluindo upsert e pull incremental, sobre conjuntos de dados no HDFS.

NOTA

Para usar o Hudi, verifique se o serviço Spark2x foi instalado no cluster do MRS.

Figura 6-59 Arquitetura básica de Hudi



Características

- O recurso de transação ACID suporta importação de dados em tempo real para o lake e importação de dados em lote para o data lake.
- Múltiplas capacidades de visualização (visualização otimizada para leitura/visualização incremental/visualização em tempo real) permitem uma análise rápida dos dados.
- O design de controle de simultaneidade multi-versão (MVCC) suporta rastreamento inverso de versão de dados.
- O gerenciamento automático de tamanhos de arquivos e layouts otimiza o desempenho da consulta e fornece dados quase em tempo real para consultas.
- Leitura e gravação simultâneas são suportadas. Os dados podem ser lidos ao serem gravados com base no isolamento de snapshot.
- Inicialização é suportada para converter tabelas existentes em conjuntos de dados de Hudi.

Principais tecnologias e vantagens

- Mecanismo de índice plugável: o Hudi fornece vários mecanismos de índice para atualizar e excluir rapidamente dados em massa.
- Apoio ao ecossistema: o Hudi suporta vários mecanismos de dados, incluindo o Hive, o Spark, e o Flink.

Dois tipos de tabelas apoiadas por Hudi

- Cópia em gravação

  As tabelas de cópia em gravação também são chamadas de tabelas COW. Os arquivos Parquet são usados para armazenar dados, e as operações de atualização internas precisam ser executadas reescrevendo os arquivos Parquet originais.

  - Vantagens: é eficiente porque apenas um arquivo de dados na partição correspondente precisa ser lido.
  - Desvantagens: durante a gravação de dados, uma cópia anterior precisa ser copiada e, em seguida, um novo arquivo de dados é gerado com base na cópia anterior. Este

processo é demorado. Portanto, os dados lidos pela solicitação de leitura ficam atrasados.

- Mesclagem na leitura

  As tabelas de mesclagem na leitura também são chamadas de tabelas MOR. A combinação de Parquet baseado em coluna e formato baseado em linha Avro é usada para armazenar dados. Os arquivos Parquet são usados para armazenar dados básicos, e os arquivos Avro (também chamados de arquivos de log) são usados para armazenar dados incrementais.

  - Vantagens: os dados são gravados no log delta primeiro e o tamanho do log delta é pequeno. Portanto, o custo de gravação é baixo.
  - Desvantagens: os arquivos precisam ser compactados periodicamente. Caso contrário, há um grande número de arquivos fragmentados. O desempenho de leitura é fraco porque os logs de delta e os arquivos de dados antigos precisam ser mesclados.

Hudi suporte três tipos de visualizações para capacidades de leitura em diferentes cenários

- Visualização de snapshot

  Fornece os dados instantâneos mais recentes da tabela de Hudi atual. Ou seja, uma vez que os dados mais recentes são gravados na tabela de Hudi, os dados recém-escritos podem ser consultados por meio dessa visão.

  Ambas as tabelas COW e MOR suportam esse recurso de visualização.

- Visualização incremental

  Fornece a capacidade de consulta incremental. Os dados incrementais após um commit especificado podem ser consultados. Essa visualização pode ser usada para puxar rapidamente dados incrementais.

  As tabelas COW suportam esse recurso de visualização. As tabelas MOR também suportam essa capacidade de visualização, mas a capacidade de visualização incremental desaparece quando a operação compacta é realizada.

- Visualização de leitura otimizada

  Fornece apenas os dados armazenados no arquivo Parquet mais recente.

  Essa visualização é diferente para as tabelas COW e MOR.

  Para tabelas COW, a capacidade de visualização é a mesma que a capacidade de visualização em tempo real. (As tabelas COW usam somente arquivos Parquet para armazenar dados.)

  Para tabelas MOR, apenas os arquivos base são acessados e os dados nas fatias de arquivo fornecidas desde a última operação compacta são fornecidos. Pode ser simplesmente entendido que esta visualização fornece apenas os dados armazenados em arquivos Parquet de tabelas MOR, e os dados em arquivos de log são ignorados. Os dados fornecidos por esta visualização podem não ser os mais recentes. No entanto, uma vez que a operação compacta é realizada em tabelas MOR, os dados de log incrementais são mesclados nos dados base. Neste caso, esta vista tem a mesma capacidade que a vista em tempo real.

6.13 Hue

6.13.1 Princípios básicos do Hue

Hue é um grupo de aplicações Web que interagem com componentes de Big Data do MRS. Ele ajuda você a navegar no HDFS, executar consultas no Hive e iniciar jobs do MapReduce. Hue possui aplicações que interagem com todos os componentes de Big Data do MRS.

O Hue fornece as funções do navegador de arquivos e do editor de consultas:

- O navegador de arquivos permite que você navegue e opere diretamente diferentes diretórios HDFS na GUI.
- O editor de consultas pode escrever instruções SQL simples para consultar dados armazenados no Hadoop, por exemplo, HDFS, HBase e Hive. Com o editor de consultas, você pode facilmente criar, gerenciar e executar instruções SQL e baixar os resultados da execução como um arquivo do Excel.

Na WebUI fornecida pelo Hue, você pode executar as seguintes operações nos componentes:

- HDFS:
  - Visualizar, criar, gerenciar, renomear, mover e excluir arquivos ou diretórios.
  - Upload e download de arquivos
  - Pesquisar arquivos, diretórios, proprietários de arquivos e grupos de usuários; alterar os proprietários e permissões dos arquivos e diretórios.
  - Configurar manualmente as políticas de armazenamento de diretório de HDFS e as políticas de armazenamento dinâmico.
- Hive:
  - Editar e executar instruções SQL/HQL. Salvar, copiar e editar o modelo SQL/HQL. Explicar instruções SQL/HQL. Salvar a instrução SQL/HQL e consultá-la.
  - Apresentação do banco de dados e apresentação da tabela de dados
  - Suportar a diferentes tipos de armazenamento de Hadoop
  - Use o MetaStore para adicionar, excluir, modificar e consultar bancos de dados, tabelas e exibições.

    NOTA

    Se o Internet Explorer for usado para acessar a página de Hue para executar instruções HiveSQL, a execução falhará porque o navegador tem problemas funcionais. É aconselhável usar um navegador compatível, por exemplo, o Google Chrome.
- Impala:
  - Editar e executar instruções SQL/HQL. Salvar, copiar e editar o modelo SQL/HQL. Explicar instruções SQL/HQL. Salvar a instrução SQL/HQL e consultá-la.
  - Apresentação do banco de dados e apresentação da tabela de dados
  - Suportar a diferentes tipos de armazenamento de Hadoop
  - Use o MetaStore para adicionar, excluir, modificar e consultar bancos de dados, tabelas e exibições.

    NOTA

    Se o Internet Explorer for usado para acessar a página de Hue para executar instruções HiveSQL, a execução falhará porque o navegador tem problemas funcionais. Você é aconselhado a usar um navegador compatível, por exemplo, o Google Chrome.
- MapReduce: verifique as tarefas do MapReduce que estão sendo executadas ou foram concluídas nos clusters, incluindo status, horário de início e término e logs de execução.

- Oozie: Hue fornece a função de gerenciador de jobs Oozie, neste caso, você pode usar Oozie no modo GUI.
- ZooKeeper: Hue fornece a função do navegador ZooKeeper para você usar o ZooKeeper no modo GUI.

Para detalhes sobre a Hue, visite **https://gethue.com/**.

Arquitetura

Hue, adotando o design MTV (Model-Template-View), é um programa de aplicação web rodando em Django Python. (Django Python é uma estrutura de aplicações Web que utiliza código-fonte aberto.)

Hue consiste em Supervisor Process e WebServer. O Supervisor Process é o principal processo de Hue que gerencia os processos de aplicação. O Supervisor Process e o WebServer interagem com aplicações no WebServer por meio de APIs Thrift/REST, como mostrado em **Figura 6-60**.

Figura 6-60 Arquitetura do Hue

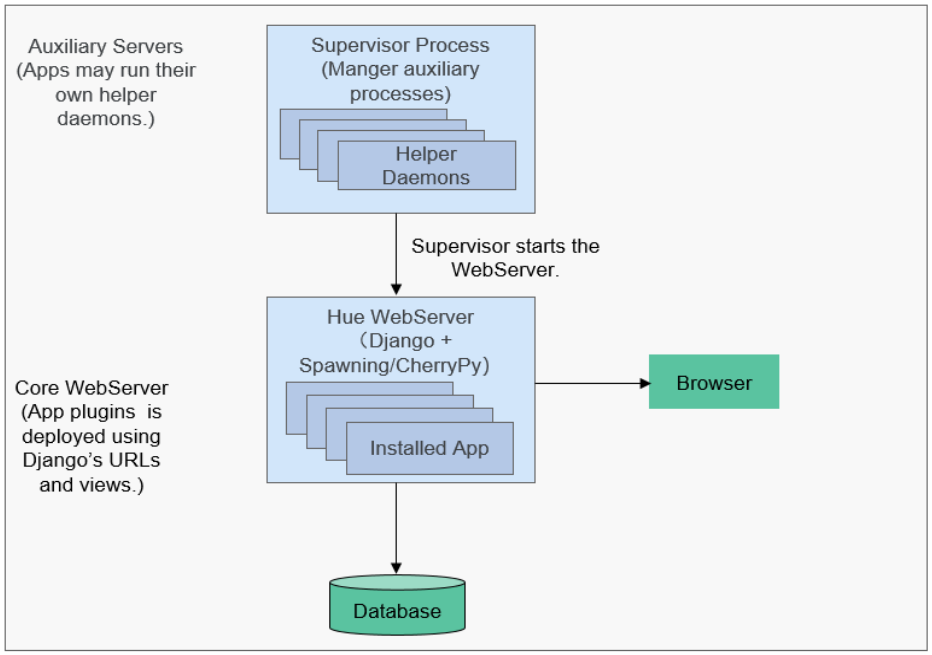


Tabela 6-14 descreve os componentes mostrados em **Figura 6-60**.

Tabela 6-14 Descrição da arquitetura

Nome da conexão	Descrição
Supervisor Process	Gerencia processos de aplicações do WebServer, como iniciar, parar e monitorar os processos.

Nome da conexão	Descrição
Hue WebServer	Fornece as seguintes funções através da estrutura da Web de Django Python: ● Implementa aplicações. ● Fornece a GUI. ● Conecta-se a bancos de dados para armazenar dados persistentes de aplicações.

6.13.2 Relação entre Hue e outros componentes

Relação entre Hue e clusters do Hadoop

Figura 6-61 mostra como Hue interage com clusters do Hadoop.

Figura 6-61 Hue e clusters do Hadoop

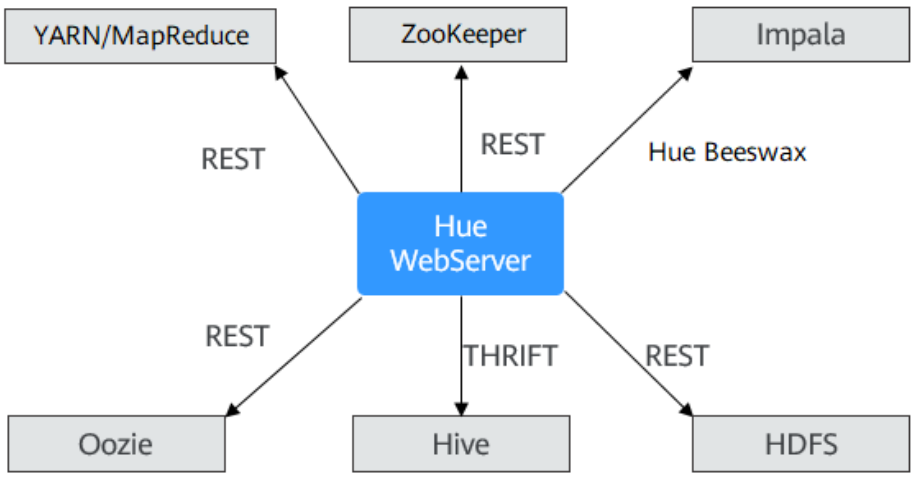


Tabela 6-15 Relação entre Hue e outros componentes

Nome da conexão	Descrição
HDFS	O HDFS fornece APIs REST para interagir com o Hue para consultar e operar arquivos HDFS. O Hue empacota uma solicitação do usuário em dados de interface, envia a solicitação para o HDFS por meio de APIs REST e exibe os resultados da execução na IU da Web.
Hive	O Hive fornece interfaces do Thrift para interagir com o Hue, executar instruções SQL do Hive e consultar metadados da tabela. Se você editar instruções HQL na IU da Web do Hue, o Hue enviará as instruções HQL ao servidor do Hive por meio das APIs do Thrift e exibirá os resultados da execução na IU da Web.

Nome da conexão	Descrição
YARN/MapReduce	O MapReduce fornece APIs REST para interagir com o Hue e consultar informações de jobs do YARN. Se você for para a IU da Web Hue, insira os parâmetros de filtro, a IU envia os parâmetros para o plano de fundo e Hue chama as APIs REST fornecidas pelo MapReduce para obter informações como o status da execução da tarefa, a hora de início/fim, o log de execução e muito mais.
Oozie	Oozie fornece APIs REST para interagir com o Hue, criar fluxos de trabalho, coordenadores e pacotes e gerenciar e monitorar tarefas. Um fluxo de trabalho gráfico, um coordenador e um editor de pacotes são fornecidos na IU da Web do Hue. O Hue invoca as APIs REST do Oozie para criar, modificar, excluir, enviar e monitorar fluxos de trabalho, coordenadores e pacotes.
ZooKeeper	O ZooKeeper fornece APIs REST para interagir com o Hue e consultar informações de nó do ZooKeeper. As informações do nó do ZooKeeper são exibidas na IU da Web do Hue. Hue invoca as APIs REST do ZooKeeper para obter as informações do nó.
Impala	O Impala fornece APIs do Hue Beeswax para interagir com o Hue, executar instruções SQL do Hive e consultar metadados de tabelas. Se você editar instruções HQL na IU da Web do Hue, o Hue enviará as instruções HQL para o servidor do Hive por meio das APIs do Hue Beeswax e exibirá os resultados da execução na IU da Web.

6.13.3 Recursos de código aberto aprimorados do Hue

Recursos de código aberto aprimorados do Hue

- Política de armazenamento: o número de cópias de arquivos do HDFS varia dependendo da mídia de armazenamento. Esse recurso permite que você defina manualmente uma política de armazenamento de diretório do HDFS ou pode ajustar automaticamente a política de armazenamento de arquivos, modificar o número de cópias de arquivos, mover o diretório de arquivos e excluir arquivos com base no último tempo de acesso e tempo de modificação de arquivos do HDFS para utilizar totalmente a capacidade de armazenamento e melhorar o desempenho do armazenamento.
- Motor do MR: você pode usar o mecanismo MapReduce para executar instruções SQL do Hive.
- Aprimoramento da confiabilidade: é implementado no modo ativo/em espera. Ao interconectar-se com HDFS, Oozie, Hive e YARN, o Hue pode trabalhar no modo de failover ou balanceamento de carga.

6.14 Impala

Impala

O Impala fornece consultas SQL rápidas e interativas diretamente nos dados do Apache Hadoop armazenados no HDFS, HBase ou no Object Storage Service (OBS). Além de usar a mesma plataforma de armazenamento unificada, o Impala também usa os mesmos metadados, sintaxe SQL (Hive SQL), driver ODBC e interface de usuário (UI de consulta de Impala no Hue) como Apache Hive. Isso fornece uma plataforma familiar e unificada para consultas em tempo real ou orientadas a lotes. Impala é uma adição às ferramentas disponíveis para consultar Big Data. O Impala não substitui as estruturas de processamento em lote construídas em MapReduce como o Hive. Hive e outras estruturas construídas no MapReduce são mais adequados para jobs em lote de longa duração.

O Impala oferece os seguintes recursos:

- Recursos de SQL-92 mais comuns do Hive Query Language (HiveQL), incluindo SELECT, JOIN e funções agregadas
- Armazenamento HDFS, HBase e OBS, incluindo:
  - Formatos de arquivo HDFS: arquivos de texto delimitados, Parquet, Avro SequenceFile e RCFile
  - Codecs de compressão: Snappy, GZIP, Deflate, BZIP
- Interfaces comuns de acesso a dados, incluindo:
  - Driver JDBC
  - Driver ODBC
  - Hue Beeswax e a interface de consulta de Impala
- Interface de linha de comando **impala-shell**
- Autenticação de Kerberos

Impala aplica-se a análise off-line (como log e análise de status de cluster) de consultas de dados em tempo real, mineração de dados em larga escala (como análise do comportamento do usuário, análise da região de interesse e exibição da região) e outros cenários.

Para mais informações sobre o Impala, visite **https://impala.apache.org/impala-docs.html**.

Impala consiste em três papéis: Impala Daemon (Impalad), Impala StateStore e Serviço de catálogo do Impala.

Impala Daemon

O componente principal do Impala é o Impala daemon, fisicamente representado pelo processo **impalad**.

Algumas das principais funções que um Impala daemon executa são:

- Executa em todos os nós de dados.
- Lê e grava em arquivos de dados.
- Aceita consultas transmitidas do comando **impala-shell**, Hue, JDBC ou ODBC.

- Paralela às consultas e transmite os resultados da consulta intermediária de volta ao coordenador central.
- Invoca um nó para retornar os resultados da consulta ao cliente.

Os Impala daemons estão em constante comunicação com o StateStore para confirmar quais daemons estão íntegros e podem aceitar novos trabalhos.

Impala StateStore

O componente do Impala conhecido como StateStore verifica a integridade de todos os Impala daemons em um cluster e continuamente retransmite suas descobertas para cada um desses daemons. Ele é fisicamente representado por um processo de daemon chamado **statestored**. Você só precisa de tal processo em um host em um cluster. Se um Impala daemon ficar off-line devido à falha de hardware, erro de rede, problema de software ou outro motivo, o StateStore informa todos os outros Impala daemons para que futuras consultas possam evitar solicitações ao Impala daemon inacessível.

Serviço de catálogo do Impala

O componente do Impala conhecido como Serviço de catálogo retransmite as alterações de metadados das instruções SQL do Impala para todos os Impala daemons em um cluster. É fisicamente representado por um processo de daemon chamado **catalogd**. Quando você cria uma tabela, carrega dados, e assim por diante através do Hive, você precisa emitir REFRESH ou INVALIDATE METADATA em um Impala daemon antes de executar uma consulta lá. O serviço de catálogo evita a necessidade de emitir declarações REFRESH e INVALIDATE METADATA quando as alterações de metadados são realizadas por instruções emitidas através do 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

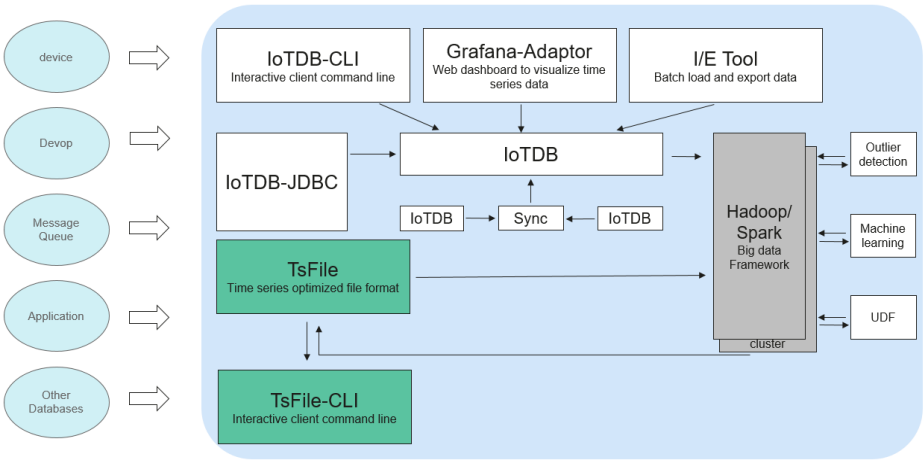
This section applies to MRS 3.1.5 or later.

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

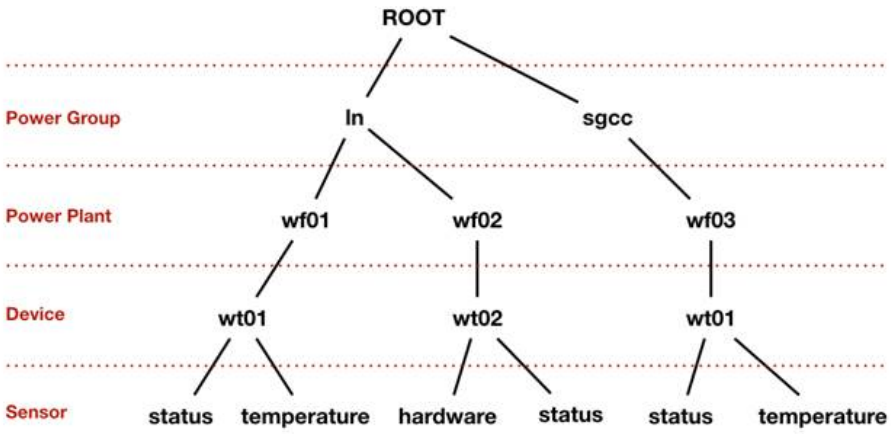


- Users can use Java Database Connectivity (JDBC) to import the time series data and system status data (such as server load, CPU usage and memory usage) collected from device sensors, as well as time series data in message queues, applications, or other databases, to the local or remote IoTDB. Users 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 Princípios básicos de Kafka

Kafka é um serviço de log de commit de código aberto, distribuído, particionado e replicado. Kafka é uma mensagem de publicação-assinatura, repensada como um log de commit distribuído. Ele fornece recursos semelhantes ao Serviço de Mensagens do Java (JMS), mas outro design. Possui resistência a mensagens, alta taxa de transferência, métodos distribuídos, suporte a vários clientes e tempo real. Aplica-se ao consumo de mensagens on-line e off-line, como coleta regular de mensagens, rastreamento de atividade do site, agregação de dados de operação do sistema estatístico (dados de monitoramento) e coleta de logs. Esses cenários envolvem grandes quantidades de coleta de dados para serviços de Internet.

Estrutura de Kafka

Os produtores publicam dados sobre os tópicos e os consumidores assinam os tópicos e consomem mensagens. Um broker é um servidor em um cluster do Kafka. Para cada tópico, o cluster do Kafka mantém partições para escalabilidade, paralelismo e tolerância a falhas. Cada partição é uma sequência ordenada e imutável de mensagens que é continuamente anexada a - um log de commit. A cada mensagem em uma partição é atribuído um ID sequencial, que é chamado deslocamento.

Figura 6-64 Arquitetura de Kafka

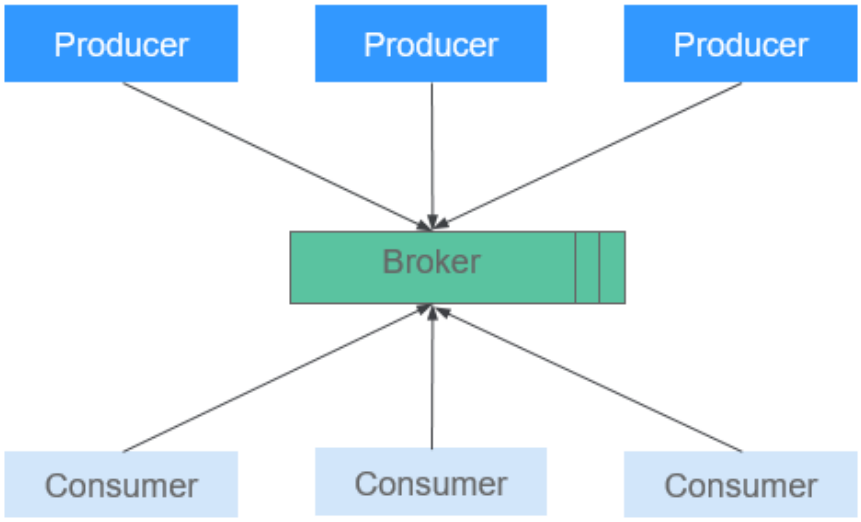


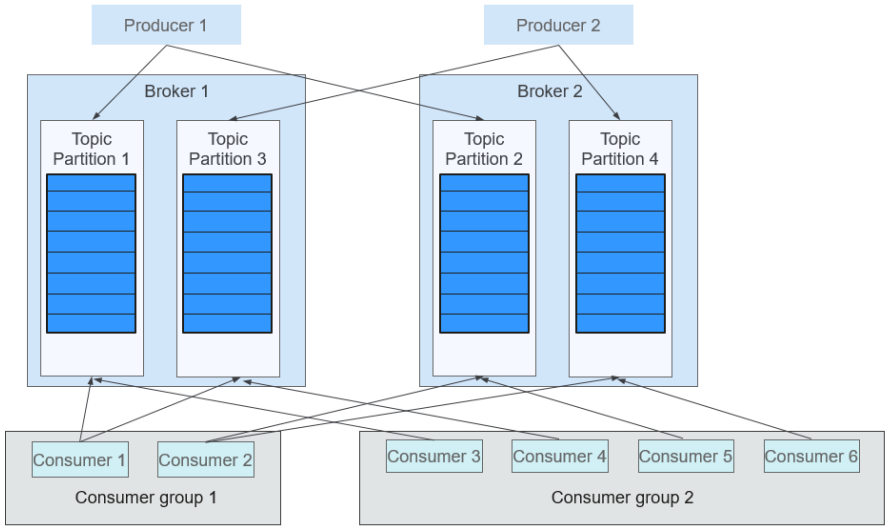
Tabela 6-16 Descrição da arquitetura de Kafka

Nome	Descrição
Broker	Um broker é um servidor em um cluster do Kafka.
Tópico	Um tópico é uma categoria ou nome do feed no qual as mensagens são publicadas. Um tópico pode ser dividido em várias partições, que podem atuar como uma unidade paralela.

Nome	Descrição
Partição	Uma partição é uma sequência ordenada e imutável de mensagens que é continuamente anexada a - um log de commit. As mensagens nas partições recebem um número de identificação sequencial chamado offset que identifica exclusivamente cada mensagem dentro da partição.
Produtor	Produtores publicam mensagens em um tópico do Kafka.
Consumidor	Os consumidores assinam tópicos e processam o feed de mensagens publicadas.

Figura 6-65 mostra as relações entre os módulos.

Figura 6-65 Relações entre os módulos de Kafka



Os consumidores se rotulam com um nome de grupo de consumidores e cada mensagem publicada em um tópico é entregue a uma instância de consumidor dentro de cada grupo de consumidores inscrito. Se todas as instâncias de consumidores pertencerem ao mesmo grupo de consumidores, as cargas serão distribuídas uniformemente entre os consumidores. Como mostrado na figura anterior, Consumidor1 e Consumidor2 trabalham no modo de compartilhamento de carga; Consumidor3, Consumidor4, Consumidor5 e Consumidor6 trabalham no modo de compartilhamento de carga. Se todas as instâncias de consumidores pertencem a diferentes grupos de consumidores, as mensagens são transmitidas para todos os consumidores. Conforme mostrado na figura anterior, as mensagens no Tópico 1 são transmitidas para todos os consumidores no Grupo de Consumidores1 e no Grupo de Consumidores2.

Para detalhes sobre a arquitetura e os princípios de Kafka, veja **https://kafka.apache.org/24/documentation.html**.

Princípio

- **Confiabilidade da mensagem**

  Quando um broker de Kafka recebe uma mensagem, ele armazena a mensagem em um disco persistentemente. Cada partição de um tópico tem várias réplicas armazenadas em diferentes nós do broker. Se um nó estiver com defeito, as réplicas em outros nós poderão ser usadas.

- **Alta taxa de transferência**

  Kafka fornece alta taxa de transferência das seguintes maneiras:

  - As mensagens são gravadas em discos em vez de serem armazenadas em cache na memória, utilizando totalmente o desempenho sequencial de leitura e gravação dos discos.
  - O uso de cópia zero elimina as operações de I/O.
  - Os dados são enviados em lotes, melhorando a utilização da rede.
  - Cada tópico é dividido em várias partições, o que aumenta o processamento simultâneo. Operações simultâneas de leitura e gravação podem ser realizadas entre vários produtores e consumidores. Os produtores enviam mensagens para partições especificadas com base no algoritmo usado.

- **Mecanismo de assinatura-notificação de mensagem**

  Os consumidores se inscrevem em tópicos interessados e consomem dados no modo pull. Os consumidores podem escolher o modo de consumo, como consumo em lote, consumo repetido e consumo a partir do final, e controlar a velocidade de extração da mensagem com base na situação real. Os consumidores precisam manter os registros de consumo por si mesmos.

- **Escalabilidade**

  Quando nós de broker são adicionados para expandir a capacidade do cluster de Kafka, os brokers recém-adicionados se registram no ZooKeeper. Depois que o registro é bem sucedido, os procedimentos e os consumidores podem sentir a mudança em tempo hábil e fazer ajustes relacionados.

Recursos de código aberto

- Confiabilidade

  Métodos de processamento de mensagens como **At-Least Once**, **At-Most Once** e **Exactly Once** são fornecidos. O status de processamento de mensagens é mantido pelos consumidores. Kafka precisa trabalhar com a camada de aplicação para implementar **Exactly Once**.

- Alta taxa de transferência

  Alta taxa de transferência é fornecida para publicação e assinatura de mensagens.

- Persistência

  As mensagens são armazenadas em discos e podem ser usadas para consumo em lotes e programas de aplicações em tempo real. A persistência e a replicação de dados evitam a perda de dados.

- Distribuição

  Um sistema distribuído é fácil de ser expandido externamente. Todos os produtores, brokers e consumidores suportam a implementação de vários clusters distribuídos. Os sistemas podem ser dimensionados sem interromper a execução do software ou desligar as máquinas.

UI do Kafka

A interface do usuário do Kafka fornece serviços da Web do Kafka, exibe informações básicas sobre módulos funcionais, como brokers, tópicos, partições e consumidores em um cluster do Kafka e fornece entradas de operação para comandos comuns do Kafka. A interface do usuário do Kafka substitui o Kafka Manager para fornecer serviços da Web seguros do Kafka que atendam às especificações de segurança.

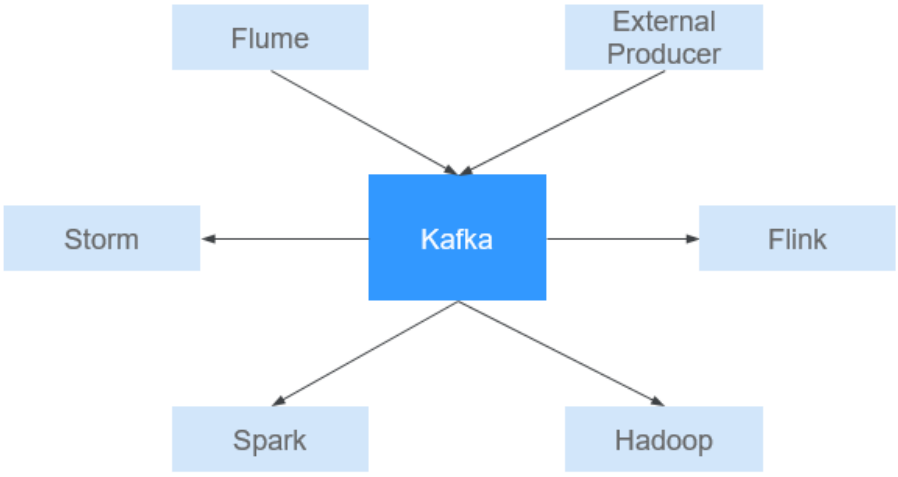
Você pode executar as seguintes operações na interface do usuário do Kafka:

- Verifique o status do cluster (tópicos, consumidores, deslocamentos, partições, réplicas e nós).
- Redistribua partições no cluster.
- Crie um tópico com configurações de tópicos opcionais.
- Exclua um tópico (suportado quando **delete.topic.enable** é definido como **true** para o serviço Kafka).
- Adicione partições a um tópico existente.
- Atualize configurações para um tópico existente.
- Opcionalmente, ative a sondagem JMX para métricas no nível do broker e no nível do tópico.

6.16.2 Relação entre Kafka e outros componentes

Sendo um sistema de publicação e assinatura de mensagens, o Kafka fornece métodos de transmissão de dados de alta velocidade entre diferentes subsistemas da plataforma FusionInsight. Ele pode receber mensagens externas em tempo real e fornece as mensagens para os serviços on-line e off-line para processamento. A figura a seguir mostra a relação entre Kafka e outros componentes.

Figura 6-66 Relação com outros componentes



6.16.3 Recursos de código aberto aprimorados do Kafka

Recursos de código aberto aprimorados do Kafka

- Monitora as seguintes métricas em nível de tópico:
  - Tráfego de entrada de tópico
  - Tráfego de saída de tópico
  - Tráfego rejeitado de tópico
  - Número de solicitações de busca com falha por segundo
  - Número de solicitações de produção com falha por segundo
  - Número de mensagens de entrada de tópicos por segundo
  - Número de solicitações de busca por segundo
  - Número de solicitações de produção por segundo
- Consulta o mapeamento entre as IDs do broker e os endereços IP do nó. Em clientes de Linux, o **kafka-broker-info.sh** pode ser usado para consultar o mapeamento entre IDs de broker e endereços IP de nó.

6.17 KafkaManager

O KafkaManager é uma ferramenta para gerenciar o Apache Kafka e fornece monitoramento e gerenciamento de métricas baseadas em GUI de clusters do Kafka.

O KafkaManager oferece suporte às seguintes operações:

- Gerenciar vários clusters do Kafka.
- Inspeção fácil dos estados do cluster (tópicos, consumidores, deslocamentos, partições, réplicas e nós)
- Executar eleição de réplica preferida.
- Gerar atribuições de partição com a opção de selecionar corretores para usar.
- Executar a reatribuição da partição (com base nas atribuições geradas).
- Criar um tópico com configurações de tópicos opcionais (várias versões de cluster do Kafka são suportadas).
- Excluir um tópico (suportado apenas em 0.8.2+ e **delete.topic.enable=true** é definido na configuração do agente).
- Gerar em lote atribuições de partição para vários tópicos com a opção de selecionar corretores para usar.
- Executar reatribuição em lote de partições para vários tópicos.
- Adicionar partições a um tópico existente.
- Atualizar configurações para um tópico existente.
- Opcionalmente, ative a pesquisa JMX para métricas no nível do agente e no nível do tópico.
- Opcionalmente, filtra os consumidores que não tenham diretórios de ids/ owner/ &offsets/ no ZooKeeper.

6.18 KrbServer e LdapServer

6.18.1 Princípios de KrbServer e LdapServer

Visão geral

Para gerenciar as permissões de controle de acesso em dados e recursos em um cluster, recomenda-se que o cluster seja instalado no modo de segurança. No modo de segurança, uma aplicação de cliente deve ser autenticada e uma sessão segura deve ser estabelecida antes que a aplicação acesse qualquer recurso no cluster. O MRS usa o KrbServer para fornecer autenticação Kerberos para todos os componentes, implementando um mecanismo de autenticação confiável.

O LdapServer oferece suporte ao protocol LDAP (Lightweight Directory Access Protocol) e permite armazenar dados de usuários e grupos de usuários para autenticação Kerberos.

Arquitetura

A função de autenticação de segurança para logon do usuário depende do Kerberos e do LDAP.

Figura 6-67 Arquitetura de autenticação de segurança

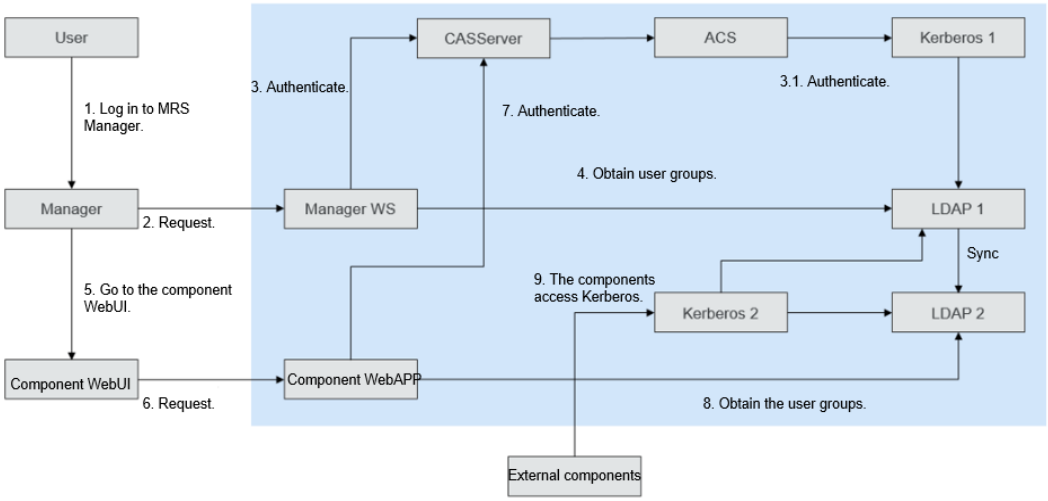


Figura 6-67 inclui três cenários:

- Efetuar logon no Manager Web UI do MRS

  A arquitetura de autenticação inclui as etapas 1, 2, 3 e 4.
- Efetuar logon em uma IU da Web de componente

  A arquitetura de autenticação inclui as etapas 5, 6, 7 e 8.
- Acessar entre componentes

  A arquitetura de autenticação inclui a etapa 9.

Tabela 6-17 Módulos principais

Nome da conexão	Descrição
Manager	Gerenciador de cluster.
Manager WS	WebBrowser
Kerberos1	Serviço KrbServer (plano de gerenciamento) implementado no Manager do MRS, ou seja, Kerberos do OMS
Kerberos2	Serviço KrbServer (plano de serviço) implementado no cluster
LDAP1	Serviço LdapServer (plano de gerenciamento) implementado no Manager do MRS, ou seja, LDAP do OMS
LDAP2	Serviço LdapServer (plano de serviço) implementado no cluster

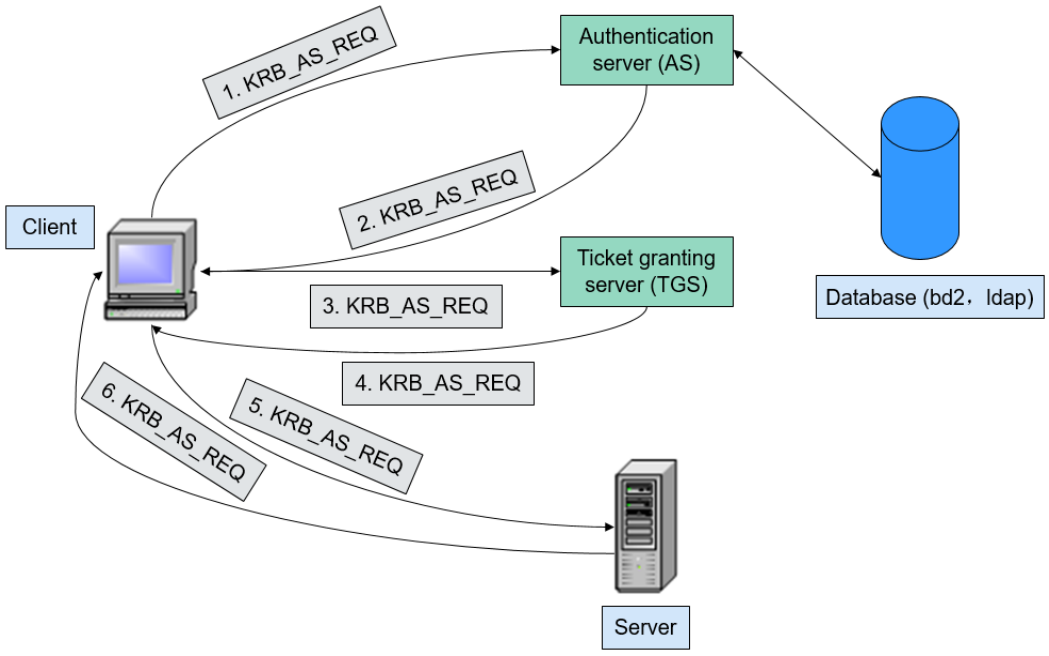
Modo de operação de dados do Kerberos1 no LDAP: as instâncias ativas e em espera do LDAP1 e as duas instâncias em espera do LDAP2 podem ser acessadas no modo de balanceamento de carga. As operações de gravação de dados podem ser executadas apenas na instância do LDAP1 ativa. As operações de leitura de dados podem ser executadas em LDAP1 ou LDAP2.

Modo de operação de dados do Kerberos2 no LDAP: as operações de leitura de dados podem ser executadas em LDAP1 e LDAP2. As operações de gravação de dados podem ser executadas apenas na instância do LDAP1 ativa.

Princípio

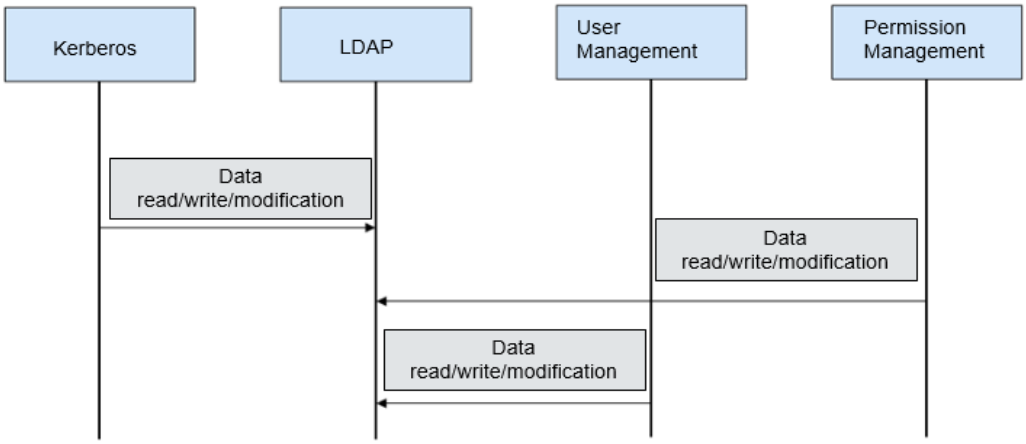
Autenticação de Kerberos

Figura 6-68 Processo de autenticação



Leitura e gravação de dados do LDAP

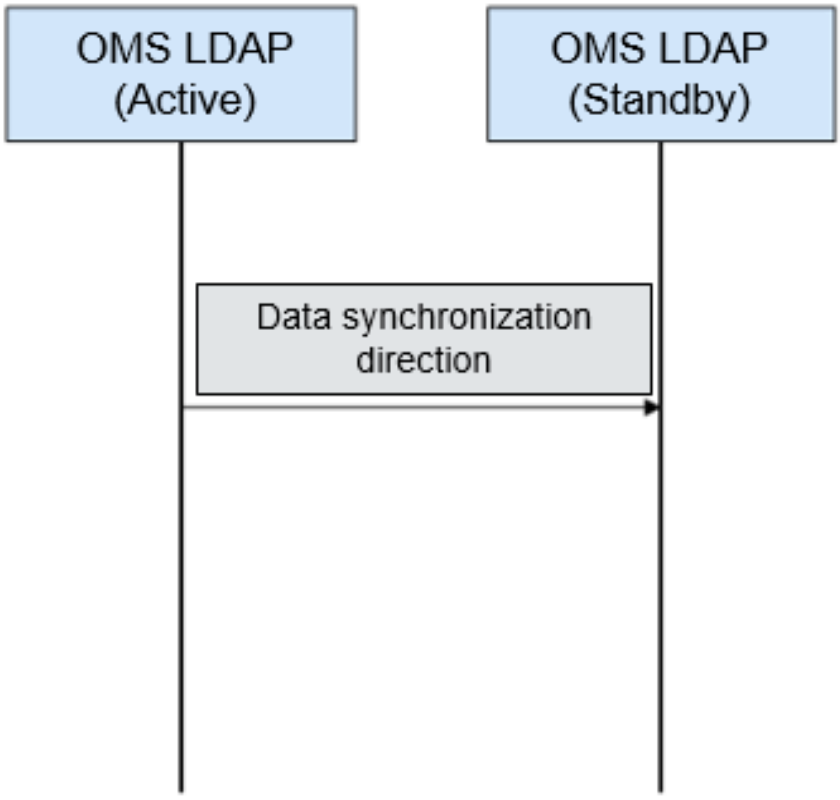
Figura 6-69 Processo de modificação de dados



Sincronização de dados do LDAP

- Sincronização de dados do LDAP do OMS antes da instalação do cluster

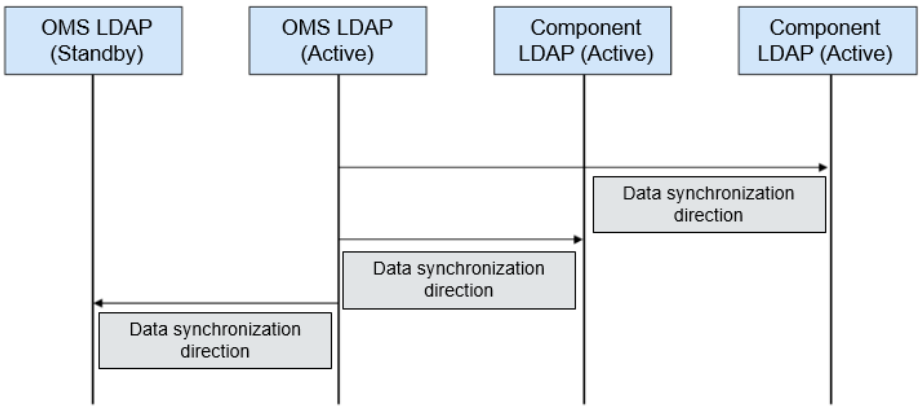
  Figura 6-70 Sincronização de dados do LDAP do OMS



  Direção de sincronização de dados antes da instalação do cluster: os dados são sincronizados do LDAP ativo do OMS para o LDAP em espera do OMS.

- Sincronização de dados do LDAP após a instalação do cluster

Figura 6-71 Sincronização de dados do LDAP



Direção da sincronização de dados após a instalação do cluster: os dados são sincronizados do LDAP do OMS ativo para o LDAP do OMS em espera, o LDAP do componente em espera e o LDAP do componente em espera.

6.18.2 Recursos de código aberto aprimorados de KrbServer e LdapServer

Recursos de código aberto aprimorados do KrbServer e LdapServer: autenticação de serviço intra-cluster

Em um cluster de MRS que usa o modo de segurança, o acesso mútuo entre serviços é implementado com base na arquitetura de segurança Kerberos. Quando um serviço (como o HDFS) no cluster deve ser iniciado, a chave de sessão correspondente (keytab, usada para autenticação de identidade da aplicação) é obtida do Kerberos. Se outro serviço (como o YARN) precisar acessar o HDFS e adicionar, excluir, modificar ou consultar dados no HDFS, o TGT e o ST correspondentes devem ser obtidos para acesso seguro.

Recursos aprimorados de código aberto do KrbServer e do LdapServer: autenticação de desenvolvimento de aplicações

Os componentes do MRS fornecem interfaces de desenvolvimento de aplicações para clientes ou clusters de produtos de serviço de camada superior. Durante o desenvolvimento de aplicações, um cluster no modo de segurança fornece interfaces de autenticação de desenvolvimento de aplicações especificadas para implementar a autenticação e o acesso à segurança de aplicações. Por exemplo, a classe UserGroupInformation fornecida pela API hadoop-common fornece várias APIs de autenticação de segurança.

- **setConfiguration()** é usado para obter configurações relacionadas e definir parâmetros como variáveis globais.
- **loginUserFromKeytab():** é usado para obter interfaces TGT.

Recursos aprimorados de código aberto do KrbServer e do LdapServer: confiança mútua entre sistemas

O MRS fornece a função de confiança mútua entre dois gerentes para implementar operações de leitura e gravação de dados entre sistemas.

6.19 Kudu

Kudu é um gerenciador de lojas de colunas desenvolvido para a plataforma Apache Hadoop. O Kudu compartilha as propriedades técnicas comuns dos aplicativos do ecossistema Hadoop, ou seja, ele é executado em hardware de mercadoria, que é escalável horizontalmente, oferecendo alta disponibilidade.

O design do Kudu tem os seguintes benefícios:

- Processamento rápido de cargas de trabalho OLAP
- Integração com o MapReduce e outros componentes do ecossistema Hadoop
- Integração apertada com o Apache Impala, tornando-o uma boa alternativa mutável ao uso do HDFS com o Apache Parquet
- Modelo de consistência forte, mas flexível, permitindo que você escolha requisitos de consistência em uma base por solicitação, incluindo a opção de consistência rigorosa-serializável
- Forte desempenho para executar cargas de trabalho sequenciais e aleatórias simultaneamente
- Gerenciamento fácil
- Os servidores Tablet e Masters de alta disponibilidade usam o Algoritmo de consenso Raft, que garante que, desde que mais da metade do número total de réplicas esteja disponível, o tablet esteja disponível para leituras e gravações. Por exemplo, se 2 de 3 réplicas ou 3 de 5 réplicas estiverem disponíveis, o tablet estará disponível. As leituras podem ser atendidas por tablets seguidores somente de leitura, mesmo em caso de falha de um tablet líder.
- Modelo de dados estruturado

Combinando todas essas propriedades, o Kudu visa o suporte para famílias de aplicativos que são difíceis ou impossíveis de implementar nas tecnologias de armazenamento do Hadoop da geração atual.

Alguns exemplos de aplicações para as quais o Kudu é uma ótima solução são:

- Aplicações de relatório em que os dados recém-chegados precisam estar imediatamente disponíveis para os usuários finais
- Aplicações de séries temporais que devem oferecer suporte simultâneo a consultas em grandes quantidades de dados históricos e consultas granulares sobre uma entidade individual que deve retornar muito rapidamente
- Aplicações que usam modelos preditivos para tomar decisões em tempo real com atualizações periódicas do modelo preditivo com base em todos os dados históricos

6.20 Loader

6.20.1 Princípios básicos do Loader

Loader é desenvolvido com base no componente de Sqoop de código aberto. Ele é usado para trocar dados e arquivos entre o MRS e bancos de dados relacionais e sistemas de arquivos. O Loader pode importar dados de bancos de dados relacionais ou servidores de arquivos para os componentes HDFS e HBase, ou exportar dados de HDFS e HBase para bancos de dados relacionais ou servidores de arquivos.

Um modelo de Loader consiste em Loader Client e Loader Server, como mostrado na **Figura 6-72**.

Figura 6-72 Modelo do Loader

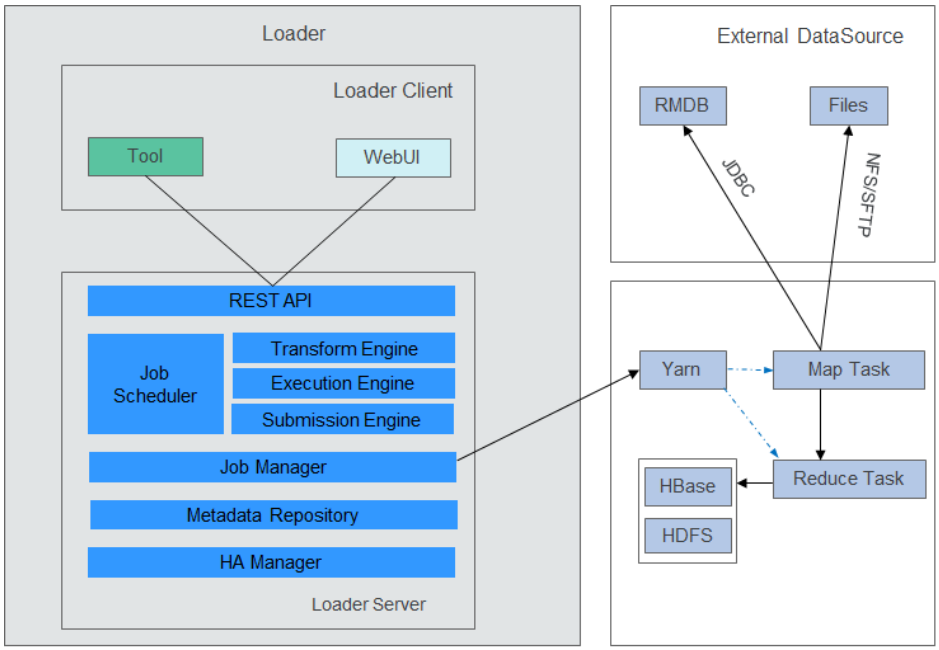


Tabela 6-18 descreve as funções de cada módulo mostrado na figura anterior.

Tabela 6-18 Componentes do modelo Loader

Módulo	Descrição
Loader Client	Cliente do Loader. Ele fornece duas interfaces: IU da Web e CLI.
Loader Server	Servidor do Loader. Processa solicitações de operação enviadas do cliente, gerencia conectores e metadados, envia jobs do MapReduce e monitora o status do job do MapReduce.
REST API	Ele fornece uma APIs Representational State Transfer (RESTful) (HTTP + JSON) para processar as solicitações de operação enviadas pelo cliente.
Job Scheduler	Agendador de jobs simples. Ele executa periodicamente jobs do Loader.
Transform Engine	Motor de transformação de dados. Ele suporta combinação de campo, corte de cordas e reversão de cordas.

Módulo	Descrição
Execution Engine	Motor de execução de job do Loader. Ele executa jobs do Loader de maneira MapReduce.
Submission Engine	Motor de envio de jobs do Loader. Ele envia jobs do Loader para MapReduce.
Job Manager	Ele gerencia jobs do Loader, incluindo criação, consulta, atualização, exclusão, ativação, desativação, início e interrupção de jobs.
Metadata Repository	Repositório de metadados. Ele armazena e gerencia dados sobre conectores do Loader, procedimentos de transformação e jobs.
HA Manager	Ele gerencia o status ativo/em espera dos processos do Loader Server. O Loader Server tem dois nós que são implementados no modo ativo/em espera.

O carregador importa ou exporta jobs em paralelo usando jobs do MapReduce. Alguns jobs de importação ou exportação podem envolver apenas as operações de Map, enquanto outras podem envolver as operações de Map e Reduce.

Loader implementa tolerância a falhas usando MapReduce. Jobs podem ser reprogramados em caso de falha na execução de trabalhos.

- **Importar dados para o HBase**

  Quando a operação Mapa é executada para jobs do MapReduce, o Loader obtém dados de uma fonte de dados externa.

  Quando uma operação de Redução é executada para um job do MapReduce, o Loader ativa o mesmo número de tarefas de Redução com base no número de Regions. As tarefas Reduce recebem dados das tarefas Map, geram HFiles por Region e armazenam os HFiles em um diretório temporário do HDFS.

  Quando um job do MapReduce é enviado, o Loader migra HFiles do diretório temporário para o HBase.

- **Importar dados para HDFS**

  Quando uma operação de Mapa é executada para um job do MapReduce, o Loader obtém dados de uma fonte de dados externa e exporta os dados para um diretório temporário (denominado *export directory*-**ldtmp**).

  Quando um job do MapReduce é enviado, o Loader migra os dados do diretório temporário para o de saída.

- **Exportar dados para um banco de dados relacional**

  Quando uma operação Map é executada para um job do MapReduce, o Loader obtém dados do HDFS ou HBase e insere os dados em uma tabela temporária (Tabela de preparo) através da API de Conectividade de banco de dados Java (JDBC).

  Quando um job do MapReduce é enviado, o Loader migra os dados da tabela temporária para uma tabela formal.

- **Exportar dados para um sistema de arquivos**

  Quando uma operação Map é executada para um job do MapReduce, o Loader obtém dados do HDFS ou HBase e grava os dados em um diretório temporário do servidor de arquivos.

Quando um job do MapReduce é enviado, o Loader migra os dados do diretório temporário para um diretório formal.

Para obter detalhes sobre a arquitetura e os princípios do Loader, consulte **https://sqoop.apache.org/docs/1.99.3/index.html**.

6.20.2 Relação entre Loader e outros componentes

Os componentes que interagem com o Loader incluem o HDFS, o HBase, o MapReduce e o ZooKeeper. O Loader funciona como um cliente para usar determinadas funções desses componentes, como armazenar dados em HDFS e HBase e ler dados de tabelas HDFS e HBase. Além disso, o Loader funciona como um cliente de MapReduce para importar ou exportar dados.

6.20.3 Recursos de código aberto aprimorados do Loader

Recurso de código aberto aprimorado do Loader: importação e exportação de dados

O Loader é desenvolvido com base no Sqoop. Além das funções Sqoop, o Loader possui os seguintes recursos aprimorados:

- Fornece funções de conversão de dados.
- Suporta conversão de configuração baseada em GUI.
- Importa dados de um servidor SFTP/FTP para HDFS/OBS.
- Importa dados de um servidor SFTP/FTP para uma tabela de HBase.
- Importa dados de um servidor SFTP/FTP para uma tabela de Phoenix.
- Importa dados de um servidor SFTP/FTP para uma tabela de Hive.
- Exporta dados do HDFS/OBS para um servidor SFTP/FTP.
- Exporta dados de uma tabela de HBase para um servidor SFTP/FTP.
- Exporta dados de uma tabela de Phoenix para um servidor SFTP/FTP.
- Importa dados de um banco de dados relacional para uma tabela de HBase.
- Importa dados de um banco de dados relacional para uma tabela de Phoenix.
- Importa dados de um banco de dados relacional para uma tabela de Hive.
- Exporta dados de uma tabela de HBase para um banco de dados relacional.
- Exporta dados de uma tabela de Phoenix para um banco de dados relacional.
- Importa dados de uma tabela particionada de Oracle para HDFS/OBS.
- Importa dados de uma tabela particionada de Oracle para uma tabela de HBase.
- Importa dados de uma tabela particionada de Oracle para uma tabela de Phoenix.
- Importa dados de uma tabela particionada de Oracle para uma tabela de Hive.
- Exporta dados do HDFS/OBS para uma tabela particionada de Oracle.
- Exporta dados do HBase para uma tabela particionada de Oracle.
- Exporta dados de uma tabela de Phoenix para uma tabela particionada de Oracle.
- Importa dados do HDFS para uma tabela de HBase, uma tabela de Phoenix e uma tabela de Hive no mesmo cluster.
- Exporta dados de uma tabela de HBase e de uma tabela de Phoenix para HDFS/OBS no mesmo cluster.

- Importa dados para uma tabela de HBase e uma tabela de Phoenix usando **bulkload** ou **put list**.
- Importa todos os tipos de arquivos de um servidor SFTP/FTP para o HDFS. O componente de código aberto Sqoop pode importar apenas arquivos de texto.
- Exporta todos os tipos de arquivos do HDFS/OBS para um servidor SFTP. O componente de código aberto Sqoop pode exportar apenas arquivos de texto e arquivos SequenceFile.
- Suporta conversão de formato de codificação de arquivos durante a importação e exportação de arquivos. Os formatos de codificação suportados incluem todos os formatos suportados pelo Java Development Kit (JDK).
- Mantém a estrutura de diretório original e os nomes de arquivo durante a importação e exportação de arquivos.
- Suporta combinação de arquivos durante a importação e exportação de arquivos. Por exemplo, se um grande número de arquivos deve ser importado, esses arquivos podem ser combinados em *n* arquivos (*n* pode ser configurado).
- Suporta filtragem de arquivos durante a importação e exportação de arquivos. As regras de filtragem suportam caracteres curinga e expressões regulares.
- Suporta importação e exportação em lote de tarefas ETL.
- Suporta consulta por página e palavra-chave e gerenciamento de grupo de tarefas ETL.
- Fornece endereços IP flutuantes para componentes externos.

6.21 Manager

6.21.1 Princípios básicos do Manager

Visão geral

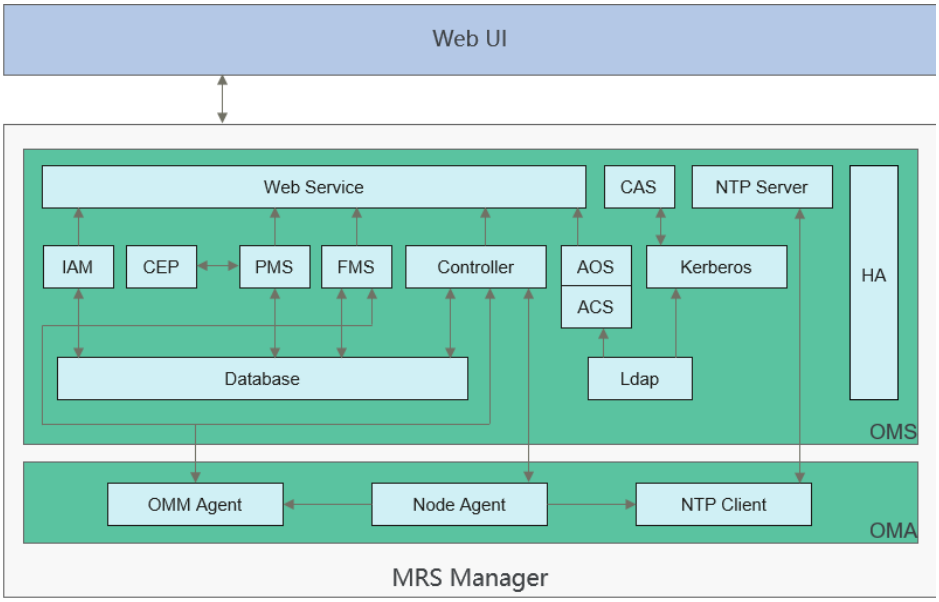
O Manager é o sistema de gerenciamento de O&M do MRS e fornece recursos unificados de gerenciamento de cluster para serviços implementados em clusters.

O Manager fornece funções como monitoramento de desempenho, alarmes, gerenciamento de usuários, gerenciamento de permissões, auditoria, gerenciamento de serviços, verificação de integridade, e coleta de registros.

Arquitetura

Figura 6-73 mostra a arquitetura lógica geral do FusionInsight Manager.

Figura 6-73 Arquitetura lógica do Manager



O gerente é composto por OMS e OMA.

- OMS: serve como nó de gerenciamento no sistema O&M. Há dois nós do OMS implantados no modo ativo/em espera.
- OMA: nó gerenciado no sistema O&M. Geralmente, existem vários nós OMA.

Tabela 6-19 descreve os módulos mostrados em **Figura 6-73**.

Tabela 6-19 Descrição do módulo de serviço

Módulo	Descrição
Serviços Web	Um serviço Web implantado no Tomcat, fornecendo API de HTTPS do Manager. Ele é usado para acessar o Manager através do navegador da Web. Além disso, ele fornece o recurso de acesso para o norte com base nos protocolos Syslog e SNMP.
OMS	Nó de gerenciamento do sistema O&M. Geralmente, existem dois nós OMS que funcionam no modo ativo/em espera.
OMA	Nó gerenciado no sistema O&M. Geralmente, existem vários nós OMA.

Módulo	Descrição
Controller	O centro de controle do gerente. Ele pode convergir informações de todos os nós do cluster e exibi-las aos administradores de cluster do MRS, bem como receber dos administradores de cluster do MRS e sincronizar informações com todos os nós do cluster de acordo com o intervalo de instruções de operação. Processo de controle do gerente. Implementa diversas ações de gestão: 1. O serviço da Web fornece várias ações de gerenciamento (como instalação, inicialização e parada de serviço e modificação de configuração) para o Controller. 2. O Controller decompõe o comando e entrega a ação para cada agente de nó, por exemplo, iniciar um serviço envolve várias funções e instâncias. 3. O Controller é responsável por monitorar a implementação de cada ação.
Node Agent	O Node Agent existe em cada nó de cluster e é um ativador do Manager em um único nó. ● O Node Agent representa todos os componentes implantados no nó para interagir com o Controller, implementando a convergência de vários nós de um cluster para um único nó. ● O Node Agent permite que o Controller execute todas as operações nos componentes implantados no nó. Ele permite que as funções do Controller sejam implementadas. O Node Agent envia mensagens de pulsação ao Controlador em um intervalo de 3 segundos. O intervalo não pode ser configurado.
IAM	Registra logs de auditoria. Cada operação sem consulta na IU do Manager tem um log de auditoria relacionado.
PMS	O módulo de monitoramento de desempenho. Ele coleta os dados de monitoramento de desempenho em cada OMA e fornece a função de consulta.
CEP	Módulo de função de convergência. Por exemplo, o espaço em disco usado de todos os OMAs é coletado como um indicador de desempenho.
FMS	Módulo de alarme. Ele coleta e consulta alarmes em cada OMA.
OMM Agent	Agente para monitoramento de desempenho e relatórios de alarme no OMA. Ele coleta dados de monitoramento de desempenho e dados de alarme no Agent Node.
CAS	Centro de autenticação unificado. Quando um usuário entra no serviço da Web, o CAS autentica o início de sessão. O navegador redireciona automaticamente o usuário para o CAS por meio de URLs.
AOS	Módulo de gerenciamento de permissões. Ele gerencia as permissões de usuários e grupos de usuários.
ACS	Módulo de gerenciamento de usuários e grupos de usuários. Ele gerencia usuários e grupos de usuários aos quais os usuários pertencem.

Módulo	Descrição
Kerberos	O LDAP é implementado no OMS e em um cluster, respectivamente. ● O OMS Kerberos fornece o logon único (SSO) e autenticação entre o Controller e o Node Agent. ● O Kerberos no cluster fornece a função de autenticação de segurança do usuário para componentes. O nome do serviço é **KrbServer** e contém duas instâncias de função: – KerberosServer:é um servidor de autenticação que fornece autenticação de segurança para o MRS. – KerberosAdmin: gerencia os processos dos usuários do Kerberos.
LDAP	O LDAP é implementado no OMS e em um cluster, respectivamente. ● O OMS LDAP fornece armazenamento de dados para autenticação do usuário. ● O LDAP no cluster funciona como backup do LDAP do OMS. O nome do serviço é **LdapServer** e a instância da função é **SlapdServer**.
Banco de dados	Banco de dados do gerenciador usado para armazenar logs e alarmes.
HA	Módulo de gerenciamento de HA que gerencia os OMSs ativos e em espera.
NTP Server NTP Client	Ele sincroniza o relógio do sistema de cada nó no cluster.

6.21.2 Principais recursos do Manager

Principais recursos: monitoramento unificado de alarme

O Manager fornece a função de monitoramento de alarme visualizada e conveniente. Os usuários podem obter rapidamente os principais indicadores de desempenho do cluster, avaliar o status de integridade do cluster, personalizar a exibição do indicador de desempenho e converter indicadores em alarmes. O Manager pode monitorar o status de funcionamento de todos os componentes e relatar alarmes em tempo real quando ocorrerem falhas. A ajuda on-line na GUI permite que você visualize contadores de desempenho e métodos de liberação de alarme para corrigir rapidamente falhas.

Principais recursos: gerenciamento unificado de permissões de usuário

O Manager fornece gerenciamento de permissões de componentes de maneira unificada.

O Manager introduz o conceito de função e usa o controle de acesso baseado em função (RBAC) para gerenciar as permissões do sistema. Ele exibe centralmente e gerencia funções de permissão dispersas de cada componente no sistema e organiza as permissões de cada componente na forma de conjuntos de permissões (papéis) para formar um conceito de permissão do sistema unificado. Ao fazê-lo, os usuários comuns não podem obter detalhes de gestão de permissões internas, e as permissões tornam-se fáceis para os administradores de

cluster do MRS gerirem, facilitando consideravelmente a gestão de permissões e melhorando a experiência do usuário.

Principais recursos: SSO

O logon único (SSO) é fornecido entre a interface do usuário da Web do Manager e a interface do usuário da Web do componente, bem como para integração entre sistemas do MRS e de terceiros.

Essa função gerencia e autentica centralmente os usuários do Manager e os usuários do componente. Todo o sistema usa LDAP para gerenciar usuários e usa Kerberos para autenticação. Um conjunto de mecanismos de gerenciamento Kerberos e LDAP são usados entre o OMS e os componentes. O SSO (incluindo logon único e saída única) é implementado por meio do CAS. Com o SSO, os usuários podem alternar facilmente as tarefas entre a interface do usuário da Web do Manager, as interfaces da Web de componentes e os sistemas de terceiros, sem alternar para outro usuário.

NOTA

- Para garantir a segurança, o servidor CAS pode reter um tíquete de concessão de tíquetes (TGT) usado por um usuário por apenas 20 minutos.
- Se um usuário não realizar nenhuma operação na página (inclusive na IU da Web do Manager e nas IUs da Web do componente) em 20 minutos, a página é bloqueada automaticamente.

Principais recursos: verificação automática de integridade e inspeção

O Manager fornece aos usuários inspeção automática em ambientes em execução do sistema e ajuda os usuários a verificar e auditar a integridade da execução do sistema por um clique, garantindo a execução correta do sistema e reduzindo os custos de operação e manutenção do sistema. Depois de visualizar os resultados da inspeção, você pode exportar relatórios para arquivamento e análise de falhas.

Principais recursos: gerenciamento de locatários

Manager introduz o conceito multi-locatário. Os recursos de CPU, memória e disco de um cluster podem ser integrados em um conjunto. O conjunto é chamado de um locatário. Um modo que envolve diferentes locatários é chamado de modo multi-locatário.

O Manager fornece a função multi-locatário, suporta um modelo de locatário baseado em níveis e permite que os locatários sejam adicionados e excluídos dinamicamente, obtendo isolamento de recursos. Como resultado, ele pode gerenciar e configurar dinamicamente os recursos de computação e os recursos de armazenamento dos locatários.

- Os recursos de computação indicam recursos de fila de tarefas do Yarn dos locatários. A cota da fila de tarefas pode ser modificada e o status e as estatísticas de uso da fila de tarefas podem ser visualizados.
- Os recursos de armazenamento podem ser armazenados no HDFS. Você pode adicionar e excluir os diretórios de armazenamento HDFS dos locatários e definir as cotas de quantidade de arquivos e o espaço de armazenamento dos diretórios.

Como uma plataforma unificada de gerenciamento de locatários do MRS, o MRS Manager permite que os usuários criem e gerenciem locatários em clusters com base nos requisitos de serviço.

- Funções, recursos de computação e recursos de armazenamento são criados automaticamente quando os locatários são criados. Por padrão, todas as permissões dos

novos recursos de computação e recursos de armazenamento são alocadas às funções de um locatário.

- Depois de modificar os recursos de computação ou armazenamento do locatário, as permissões das funções do locatário são atualizadas automaticamente.

O Manager também fornece a função de várias instâncias para que os usuários possam usar HBase, Hive ou Spark sozinhos no cenário de controle de recursos e isolamento de serviços. A função de várias instâncias está desabilitada por padrão e pode ser ativada manualmente.

Principais recursos: compatibilidade com diferentes linguagens

O Manager suporta vários idiomas e seleciona automaticamente chinês ou inglês com base na preferência de idioma do navegador. Se o idioma preferido do navegador for chinês, o Manager exibirá o portal em chinês; se o idioma preferido do navegador não for chinês, o Manager exibirá o portal em inglês. Você também pode alternar entre chinês e inglês no canto inferior esquerdo da página com base na sua preferência de idioma. (Somente o MRS 3.x e versões posteriores suportam a alternância de um clique entre chinês e inglês.)

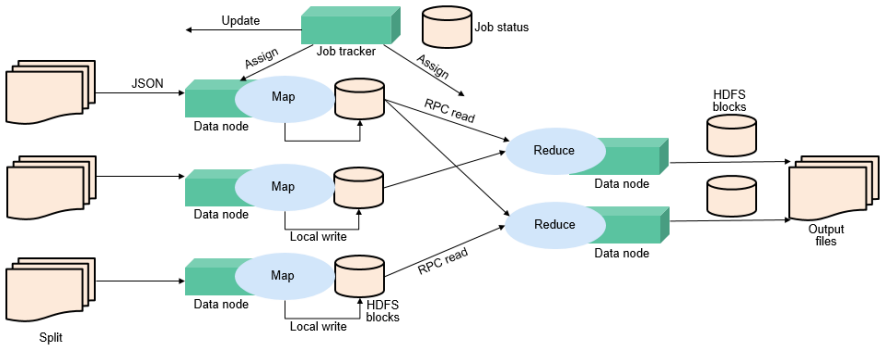
6.22 MapReduce

6.22.1 Princípios básicos do MapReduce

O MapReduce é o núcleo do Hadoop. Como uma arquitetura de software proposta pelo Google, o MapReduce é usado para computação paralela de conjuntos de dados em larga escala (maiores que 1 TB). Os conceitos "Map" e "Reduce" e seus principais pensamentos são emprestados da linguagem de programação funcional e também emprestados dos recursos da linguagem de programação vetorial.

A implementação atual do software é a seguinte: especifique uma função Map para mapear uma série de pares chave-valor em uma nova série de pares chave-valor e especifique uma função Reduce para garantir que todos os valores nos pares chave-valor mapeados compartilhem a mesma chave.

Figura 6-74 Mecanismo de processamento em lote distribuído



MapReduce é uma estrutura de software para processar grandes conjuntos de dados em paralelo. A raiz do MapReduce é Map e as funções Reduce na programação funcional. A função Map aceita um grupo de dados e o transforma em uma lista de pares chave-valor. Cada elemento no domínio de entrada corresponde a um par chave-valor. A função Reduce aceita a lista gerada pela função Map e, em seguida, reduz a lista de pares chave-valor com base nas

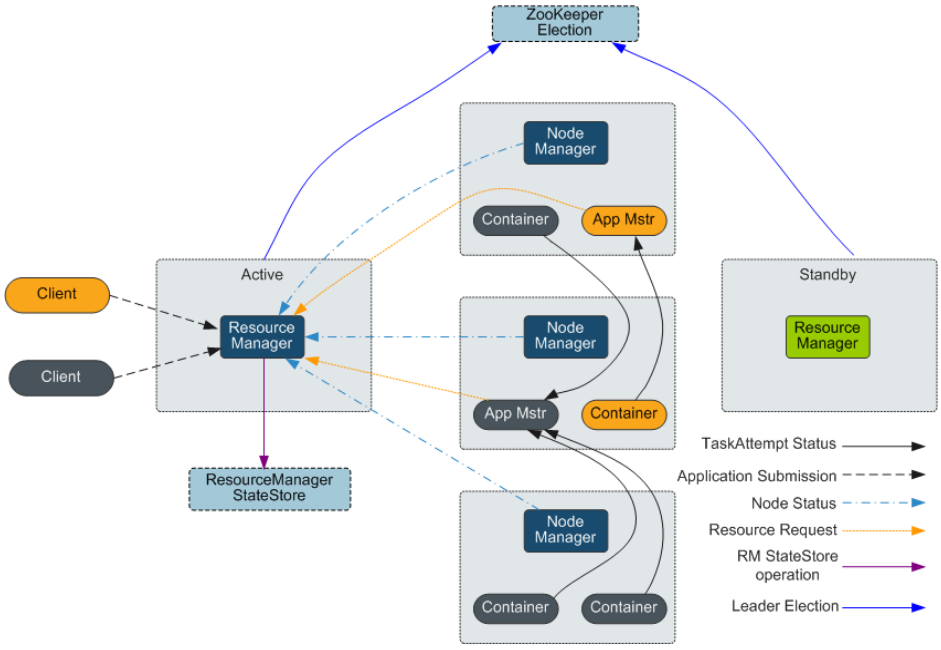
chaves. O MapReduce divide uma tarefa em várias partes e as aloca em diferentes dispositivos para processamento. Desta forma, a tarefa pode ser concluída em um ambiente distribuído em vez de um único servidor poderoso.

Para obter mais informações, consulte **Tutorial do MapReduce**.

Estrutura do MapReduce

Como mostrado em **Figura 6-75**, MapReduce é integrado ao YARN através das interfaces Cliente e ApplicationMaster do YARN e usa o YARN para solicitar recursos de computação.

Figura 6-75 Arquitetura básica do Apache YARN e MapReduce



6.22.2 Relação entre MapReduce e outros componentes

Relação entre MapReduce e HDFS

- O HDFS possui alta tolerância a falhas e alta taxa de transferência e pode ser implementado em hardware de baixo custo para armazenar dados de aplicações com conjuntos de dados massivos.
- MapReduce é um modelo de programação usado para computação paralela de grandes conjuntos de dados (maiores que 1 TB). Os dados computados pelo MapReduce vêm de várias fontes de dados, como FileSystem locais, HDFS e bancos de dados. A maioria dos dados vem do HDFS. A alta taxa de transferência do HDFS pode ser usada para ler dados massivos. Depois de ser computados, os dados podem ser armazenados no HDFS.

Relação entre MapReduce e Yarn

MapReduce é uma estrutura de computação em execução no Yarn, que é usado para processamento em lote. O MRv1 é implementado com base no MapReduce no Hadoop 1.0, que é composto por modelos de programação (novas e antigas APIs de programação), ambiente de execução (JobTracker e TaskTracker) e mecanismo de processamento de dados (MapTask e ReduceTask). Essa estrutura ainda é fraca em escalabilidade, tolerância a falhas

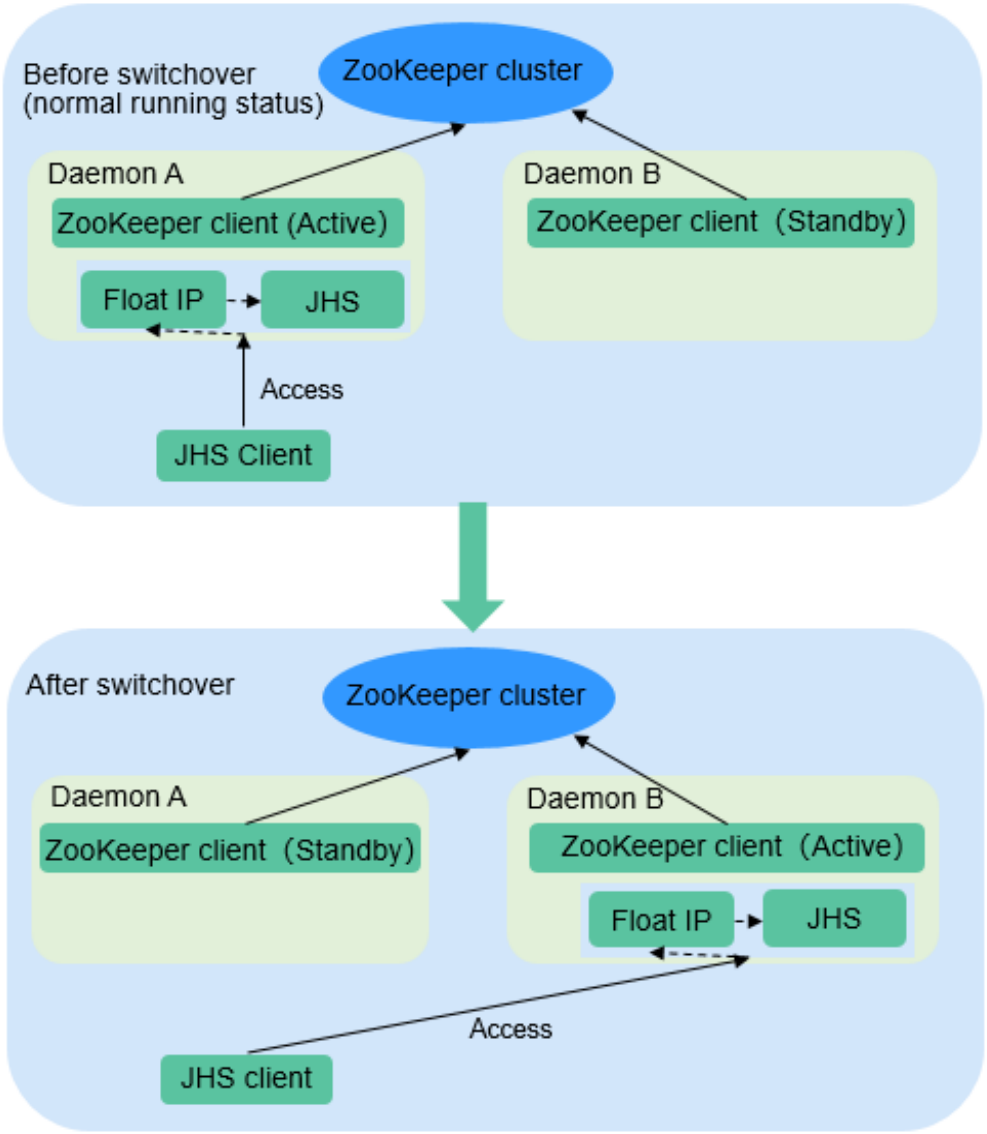
(SPOF de JobTracker) e compatibilidade com várias estruturas. (Atualmente, apenas a estrutura de computação MapReduce é suportada.) O MRv2 é implementado com base no MapReduce no Hadoop 2.0. O código-fonte reutiliza modelos de programação MRv1 e implementação do mecanismo de processamento de dados, e o ambiente de execução é composto por ResourceManager e ApplicationMaster. O ResourceManager é um novo sistema de gerenciamento de recursos, e o ApplicationMaster é responsável por cortar dados de jobs do MapReduce, atribuir tarefas, solicitar recursos, agendar tarefas e tolerar falhas.

6.22.3 Recursos de código aberto aprimorados do MapReduce

Recurso aprimorado de código aberto de MapReduce: JobHistoryServer HA

JobHistoryServer (JHS) é o servidor usado para exibir informações históricas de tarefas do MapReduce. Atualmente, o JHS de código aberto suporta apenas serviços de instância única. O JHS HA pode resolver o problema de um aplicativo não acessar a API do MapReduce quando os SPOFs ocorrem no JHS, o que faz com que o aplicativo não seja executado. Isso melhora muito a alta disponibilidade do serviço MapReduce.

Figura 6-76 Transição de status da alternância ativa/em espera de JobHistoryServer HA



JobHistoryServer de alta disponibilidade

- O ZooKeeper é usado para implementar a eleição ativa/em espera e a alternância.
- O JHS usa o endereço IP flutuante para fornecer serviços externamente.
- Os modos JHS da única instância e da implantação da HA são apoiados.
- Apenas um nó inicia o processo de JHS em um ponto de tempo para impedir que várias operações de JHS processem o mesmo arquivo.
- Você pode executar expansão, redução, migração de instâncias, atualização e verificação de integridade.

Recurso de código aberto aprimorado: melhorando o desempenho do MapReduce por otimizar o processo de mesclagem/classificação em cenários específicos

A figura abaixo mostra o fluxo de trabalho de uma tarefa do MapReduce.

Figura 6-77 Job de MapReduce

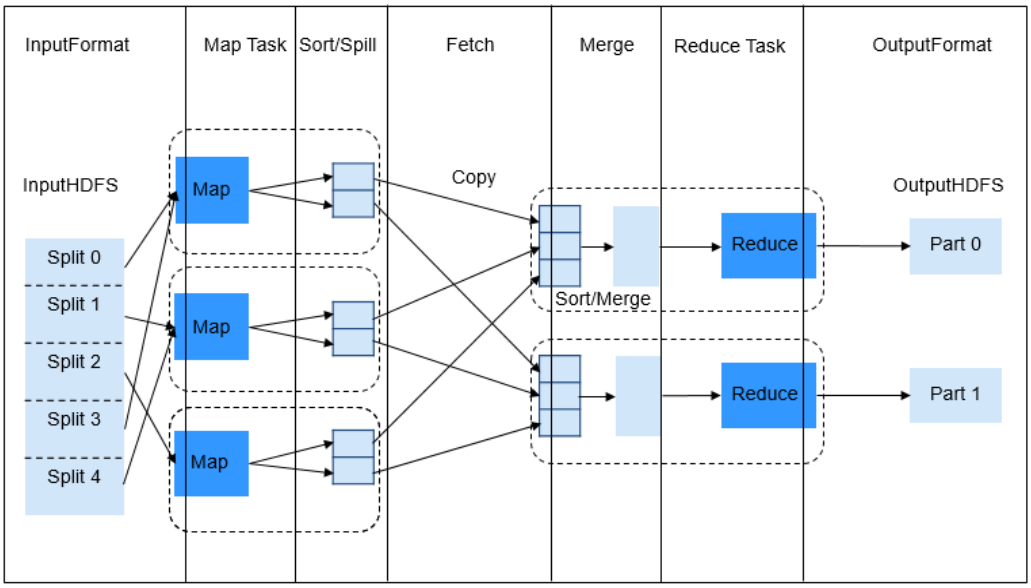
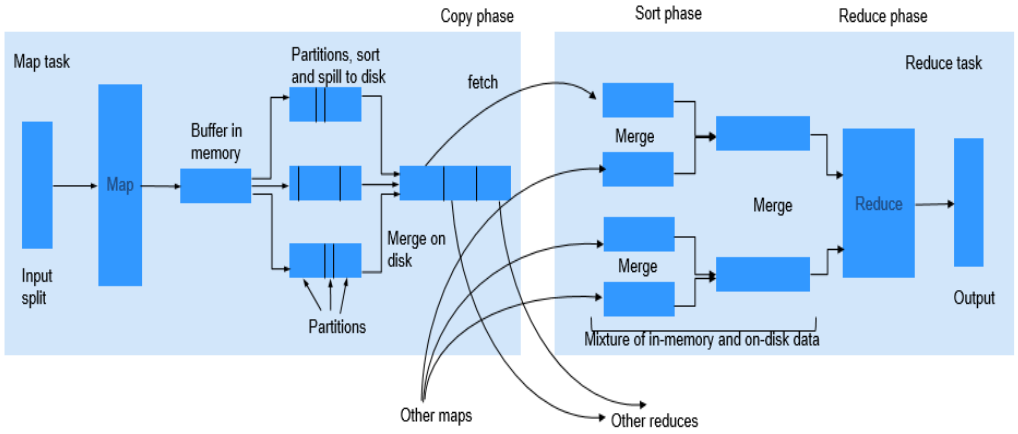


Figura 6-78 Fluxo de execução de jobs de MapReduce



O processo de Reduce é dividido em três etapas diferentes: copiar, classificar (na verdade, deveria ser chamado de mesclar) e reduzir. Na fase de copiar, o Reducer tenta buscar a saída do Maps do NodeManagers e armazená-la no Reducer na memória ou no disco. Fase de embaralhamento (classificar e mesclar), em seguida, começa. Todas as saídas de mapa obtidas estão sendo classificadas, e os segmentos de diferentes saídas de mapas são mesclados antes de serem enviados para o Reducer. Quando um job tem um grande número de mapas a serem processados, o processo de embaralhamento é demorado. Para tarefas específicas (por exemplo, tarefas de SQL como junção de hash e agregação de hash), a classificação não é obrigatória durante o processo de embaralhamento. No entanto, a classificação é necessária por padrão no processo de embaralhamento.

Esse recurso é aprimorado usando a API de MapReduce, que pode fechar automaticamente o processo de classificação para essas tarefas. Quando a classificação é desativada, a API mescla diretamente os dados de saída do Maps buscados e envia os dados para o Reducer. Isso economiza muito tempo e melhora significativamente a eficiência das tarefas de SQL.

Recurso de código aberto aprimorado: problema do arquivo pequeno de log resolvido após a otimização do MR History Server

Após a execução do job no Yarn, o NodeManager usa o LogAggregationService para coletar e enviar logs gerados ao HDFS e excluí-los do sistema de arquivos local. Depois que os logs são armazenados no HDFS, eles são gerenciados pelo MR HistoryServer. O LogAggregationService mesclará logs locais gerados por contêineres em um arquivo de log e o enviará para o HDFS, reduzindo o número de arquivos de log até certo ponto. No entanto, em um cluster de grande escala e ocupado, haverá arquivos de log excessivos no HDFS após a execução de longo prazo.

Por exemplo, se houver 20 nós, cerca de 18 milhões de arquivos de log são gerados no período de limpeza padrão (15 dias), que ocupam cerca de 18 GB da memória de um NameNode e diminuem a resposta do sistema HDFS.

Somente a leitura e a exclusão são necessárias para arquivos armazenados no HDFS. Portanto, o Hadoop Archives pode ser usado para arquivar periodicamente o diretório de arquivos de log coletados.

Arquivar logs

O módulo AggregatedLogArchiveService é adicionado ao MR HistoryServer para verificar periodicamente o número de arquivos no diretório de log. Quando o número de arquivos atinge o limite, o AggregatedLogArchiveService inicia uma tarefa de arquivamento para arquivar arquivos de log. Após o arquivamento, ele exclui os arquivos de log originais para reduzir os arquivos de log no HDFS.

Limpar logs arquivados

Hadoop Archives não suporta exclusão em arquivos arquivados. Portanto, todo o pacote de log de arquivamento deve ser excluído durante a limpeza de log. O tempo de geração de log mais recente é obtido modificando o módulo AggregatedLogDeletionService. Se todos os arquivos de log atenderem aos requisitos de limpeza, o pacote de log de arquivamento poderá ser excluído.

Navegar pelos logs arquivados

Hadoop Archives permite acesso baseado em URI ao conteúdo do arquivo no pacote de log do arquivo. Portanto, se o MR History Server detectar que os arquivos de log originais não existem durante a navegação de arquivos, ele redirecionará diretamente o URI para o pacote de log de arquivamento para acessar o arquivo de log arquivado.

NOTA

- Esta função invoca Hadoop Archives do HDFS para arquivamento de logs. Porque a execução de uma tarefa de arquivamento pelo Hadoop Archives é executar uma aplicação de MR. Portanto, depois que uma tarefa de arquivamento é executada, um registro de execução de MR é adicionado.
- Esta função de arquivamento de logs é baseada na função de coleta de logs. Por conseguinte, esta função só é válida quando a função de recolha de registos está ativada.

6.23 Oozie

6.23.1 Princípios básicos do Oozie

Introdução ao Oozie

Oozie é um mecanismo de fluxo de trabalho de código aberto usado para agendar e coordenar jobs do Hadoop.

Arquitetura

O mecanismo Oozie é uma aplicação Web integrada ao Tomcat por padrão. Oozie usa bancos de dados PostgreSQL.

O Oozie fornece um console da Web baseado em Ext, através do qual os usuários podem visualizar e monitorar os fluxos de trabalho do Oozie. O Oozie fornece uma API de serviço Web REST externa para o cliente de Oozie controlar fluxos de trabalho (como iniciar e parar operações) e orquestrar e executar tarefas de MapReduce do Hadoop. Para mais detalhes, consulte **Figura 6-79**.

Figura 6-79 Arquitetura de Oozie

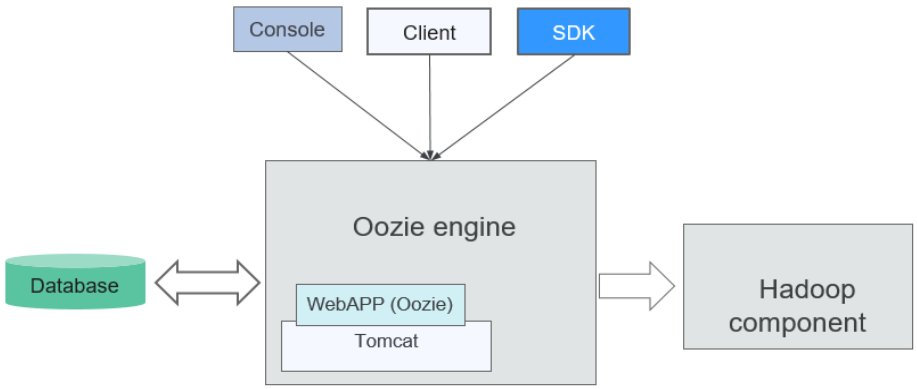


Tabela 6-20 descreve as funções de cada módulo mostrado em **Figura 6-79**.

Tabela 6-20 Descrição da arquitetura

Nome da conexão	Descrição
Console	Permite que os usuários visualizem e monitorem os fluxos de trabalho do Oozie.
Client	Controla fluxos de trabalho, incluindo enviar, iniciar, executar, plantar e restaurar fluxos de trabalho, por meio de APIs.
SDK	É a abreviação de kit de desenvolvimento de software. Um SDK é um conjunto de ferramentas de desenvolvimento usadas por engenheiros de software para estabelecer aplicações para pacotes de software específicos, estruturas de software, plataformas de hardware e sistemas operacionais.
Database	Banco de dados PostgreSQL

Nome da conexão	Descrição
WebApp (Oozie)	Funciona como o servidor Oozie. Ele pode ser implementado em um contêiner Tomcat interno ou externo. As informações registradas pelo WebApp (Oozie), incluindo logs, são armazenadas no banco de dados PostgreSQL.
Tomcat	Um servidor de aplicativos Web de código aberto gratuito
Componentes do Hadoop	Componentes subjacentes, como MapReduce e Hive, que executam os fluxos de trabalho orquestrados por Oozie.

Princípio

Oozie é um servidor de mecanismo de fluxo de trabalho que executa fluxos de trabalho do MapReduce. É também um aplicativo Web Java em execução em um contêiner de Tomcat.

Os fluxos de trabalho do Oozie são construídos usando o Hadoop Process Definition Language (HPDL). A HPDL é uma linguagem definida por XML, semelhante à JBoss jBPM Process Definition Language (jPDL). Um fluxo de trabalho do Oozie consiste no nó de controle e no nó de ação.

- O nó de controle controla a orquestração do fluxo de trabalho, como **start**, **end**, **error**, **decision**, **fork** e **join**.
- Um fluxo de trabalho do Oozie contém vários nós de ação, como MapReduce e Java.

  Todos os nós de ação são implementados e executados no modo Gráfico Acíclico Direto (DAG). Portanto, os nós de ação são executados na direção. Ou seja, o próximo nó de ação pode ser executado somente quando a execução do nó de ação anterior terminar. Quando um nó de ação termina, o servidor remoto chama de volta a interface do Oozie. Em seguida, Oozie executa o próximo nó de ação do fluxo de trabalho da mesma maneira até que todos os nós de ação sejam executados (as falhas de execução são contadas).

Os fluxos de trabalho Oozie fornecem vários tipos de nós de ação, como MapReduce, sistema de arquivos distribuídos do Hadoop (HDFS), Secure Shell (SSH), Java e subfluxos do Oozie, para dar suporte a uma ampla gama de requisitos de negócios.

6.23.2 Recursos de código aberto aprimorados do Oozie

Recurso de código aberto aprimorado: segurança melhorada

Fornece funções de administrador e usuários comuns para oferecer suporte ao gerenciamento de permissões do Oozie.

Suporta logon e logout único, acesso HTTPS e logs de auditoria.

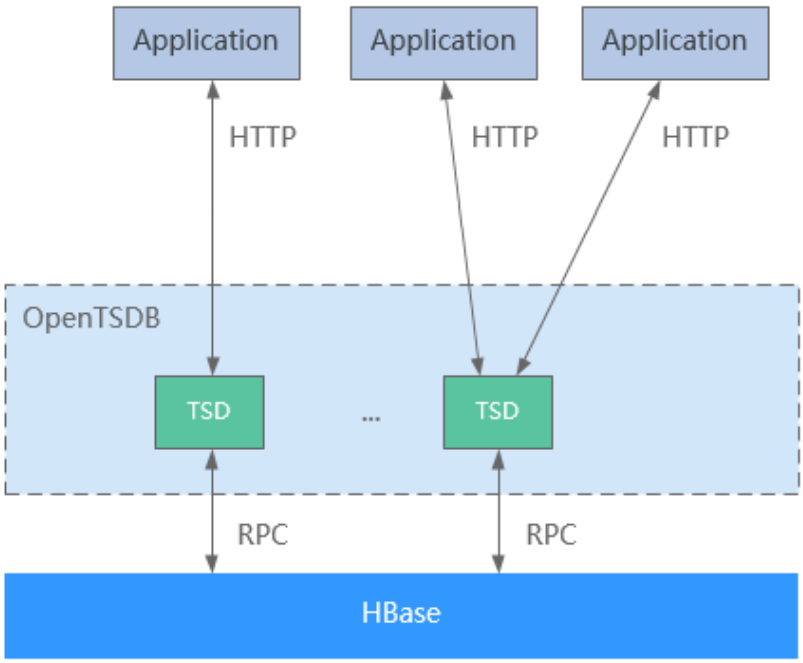
6.24 OpenTSDB

OpenTSDB é um banco de dados de séries temporais distribuído e escalável baseado em HBase. O OpenTSDB foi projetado para coletar informações de monitoramento de um cluster

de grande escala e implementar consultas de dados de segundo nível, eliminando as limitações de consultas e armazenamento de grandes quantidades de dados de monitoramento em bancos de dados comuns.

O OpenTSDB consiste em um Daemon de série temporal (TSD), bem como um conjunto de utilitários de linha de comando. A interação com o OpenTSDB é implementada principalmente executando um ou mais TSDs. Cada TSD é independente. Não há servidor mestre e nenhum estado compartilhado, então você pode executar quantos TSDs forem necessários para lidar com qualquer carga que você jogue nele. Cada TSD usa o HBase em um cluster do CloudTable para armazenar e recuperar dados de séries temporais. O esquema de dados é altamente otimizado para agregações rápidas de séries temporais semelhantes para minimizar o espaço de armazenamento. Os usuários do TSD nunca precisam acessar diretamente o armazenamento subjacente. Você pode se comunicar com o TSD através de uma API HTTP. Todas as comunicações acontecem no mesmo porto (O TSD descobre o protocolo do cliente observando os primeiros bytes que recebe).

Figura 6-80 Arquitetura de OpenTSDB



Os cenários de aplicação do OpenTSDB têm as seguintes características:

- As métricas coletadas têm um valor exclusivo em um ponto de tempo e não têm uma estrutura ou relacionamento complexo.
- As métricas de monitoramento mudam com o tempo.
- Como o HBase, o OpenTSDB apresenta alta taxa de transferência e boa escalabilidade.

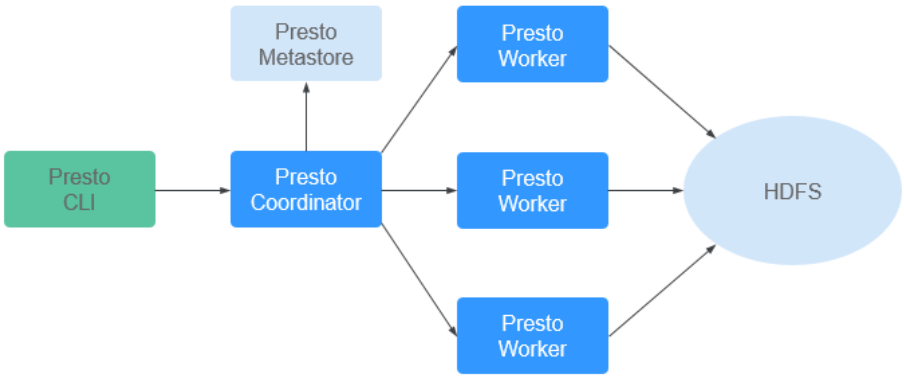
O OpenTSDB fornece uma interface de programação de aplicativos baseada em HTTP para permitir a integração com sistemas externos. Quase todos os recursos do OpenTSDB são acessíveis através da API, como consultar dados de séries temporais, gerenciar metadados e armazenar pontos de dados. Para obter detalhes, consulte **https://opentsdb.net/docs/build/html/api_http/index.html**.

6.25 Presto

O Presto é um mecanismo de consulta SQL de código aberto para executar consultas analíticas interativas em fontes de dados de todos os tamanhos. Aplica-se a análise massiva de dados estruturados/semiestruturados, agregação de dados multidimensionais/relatórios massivos, ETL, consultas ad-hoc e mais cenários.

O Presto permite consultar dados onde eles estão, incluindo HDFS, Hive, HBase, Cassandra, bancos de dados relacionais ou até mesmo armazenamentos de dados proprietários. Uma consulta Presto pode combinar diferentes fontes de dados para realizar a análise de dados entre as fontes de dados.

Figura 6-81 Arquitetura do Presto



O Presto é executado em um cluster no modo distribuído e contém um Coordinator e vários processos de Worker. As solicitações de consulta são enviadas de clientes (por exemplo, CLI) para o Coordinator. O Coordinator analisa instruções SQL, gera planos de execução e distribui os planos para vários processos de Worker para execução.

Para detalhes sobre o Presto, visite **https://prestodb.github.io/** ou **https://prestosql.io/**.

Várias instâncias de Presto

Por padrão, o MRS oferece suporte à instalação de várias instâncias do Presto para um cluster de grande escala. Ou seja, várias instâncias Worker, como Worker1, Worker2 e Worker3, são instaladas em um nó central/de tarefa. Várias instâncias Worker interagem com o Coordenador para executar tarefas de computação, melhorando significativamente a utilização de recursos do nó e a eficiência da computação.

Presto multi-instância aplica-se somente à arquitetura Arm. Atualmente, um único nó suporta no máximo quatro instâncias.

Para obter mais informações sobre implementação do Presto, consulte **https://prestodb.io/docs/current/installation/deployment.html** ou **https://trino.io/docs/current/installation/deployment.html**.

6.26 Ranger

6.26.1 Princípios básicos do Ranger

Apache Ranger oferece uma estrutura de gerenciamento de segurança centralizada e suporta autorização e auditoria unificadas. Ele gerencia o controle de acesso refinado sobre o Hadoop e componentes relacionados, como Storm, HDFS, Hive, HBase e Kafka. Você pode usar o console de interface de usuário da Web front-end fornecido pelo Ranger para configurar políticas para controlar o acesso dos usuários a esses componentes.

Figura 6-82 mostra a arquitetura do Ranger.

Figura 6-82 Estrutura do Ranger

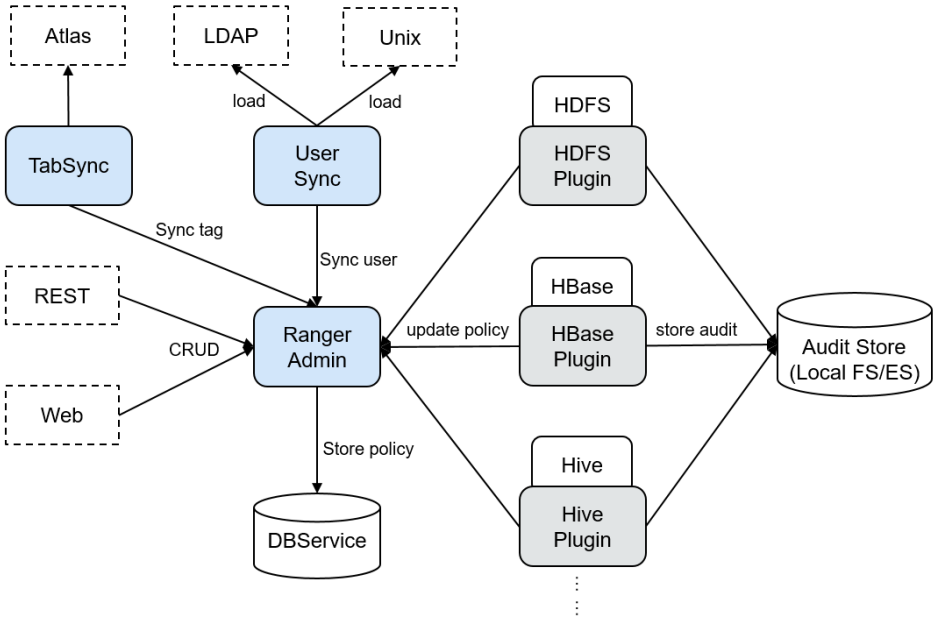


Tabela 6-21 Descrição da arquitetura

Nome da conexão	Descrição
RangerAdmin	Fornece uma WebUI e uma API RESTful para gerenciar políticas, usuários e auditoria.
UserSync	Sincroniza periodicamente informações de usuários e grupos de usuários de um sistema externo e grava as informações em RangerAdmin.
TagSync	Sincroniza periodicamente as informações de tag do serviço Atlas externo e grava as informações de tag em RangerAdmin.

Princípios do Ranger

- Plug-ins de Ranger

  O Ranger fornece plug-ins de controle de acesso baseado em políticas (PBAC) para substituir os plug-ins de autenticação originais dos componentes. Os plug-ins de Ranger são desenvolvidos com base na interface de autenticação dos componentes. Os usuários definem políticas de permissão para serviços especificados na interface da Web do Ranger. Os plug-ins de Ranger atualizam periodicamente as políticas do RangerAdmin e as armazenam em cache no arquivo local do componente. Quando uma solicitação do cliente precisa ser autenticada, o plug-in de Ranger corresponde ao usuário realizado na solicitação com a política e, em seguida, retorna uma mensagem de aceitação ou rejeição.

- Sincronização de usuário UserSync

  O UserSync sincroniza periodicamente dados de LDAP/Unix para RangerAdmin. No modo de segurança, os dados são sincronizados a partir do LDAP. No modo não-segurança, os dados são sincronizados a partir do Unix. Por padrão, o modo de sincronização incremental é usado. Em cada período de sincronização, o UserSync atualiza apenas usuários e grupos de usuários novos ou modificados. Quando um usuário ou grupo de usuários é excluído, o UserSync não sincroniza a alteração para RangerAdmin. Ou seja, o usuário ou grupo de usuários não é excluído do RangerAdmin. Para melhorar o desempenho, o UserSync não sincroniza grupos de usuários aos quais nenhum usuário pertence ao RangerAdmin.

- Auditoria unificada

  Os plug-ins de Ranger podem gravar logs de auditoria. Atualmente, os logs de auditoria podem ser armazenados em arquivos locais.

- Alta confiabilidade

  Ranger suporta dois RangerAdmins trabalhando no modo ativo/ativo. Dois RangerAdmins prestam serviços ao mesmo tempo. Se algum dos RangerAdmin estiver defeituoso, o Ranger continua a trabalhar.

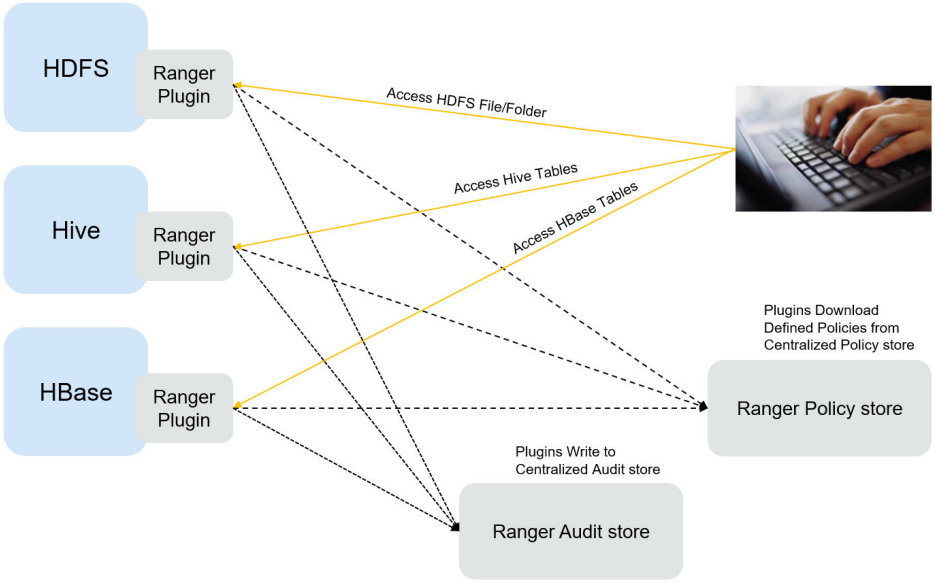
- Alto desempenho

  Ranger fornece a capacidade de balanceamento de carga. Quando um usuário acessa o Ranger WebUI usando um navegador, o Load-Balance seleciona automaticamente o RangerAdmin com a carga mais leve para fornecer serviços.

6.26.2 Relação entre Ranger e outros componentes

Ranger fornece plug-ins de autenticação baseados em PABC para que os componentes sejam executados em seus servidores. O Ranger atualmente suporta autenticação para os seguintes componentes, como HDFS, YARN, Hive, HBase, Kafka, Storm e Spark2x. Mais componentes serão suportados no futuro.

Figura 6-83 Relação entre o Ranger e outros componentes



6.27 Spark

6.27.1 Princípios básicos do Spark

NOTA

O componente Spark se aplica a versões anteriores ao MRS 3.x.

Descrição

Spark é uma estrutura de processamento de dados paralelos de código aberto. Ele ajuda você a desenvolver facilmente aplicações de Big Data unificados e realizar processamento off-line, processamento de fluxo e análise interativa de dados.

O Spark fornece uma estrutura com computação rápida, escrita e consulta interativa. O Spark tem vantagens óbvias sobre o Hadoop em termos de desempenho. O Spark usa o modo de computação na memória para evitar gargalos de I/O em cenários em que várias tarefas em um fluxo de trabalho do MapReduce processam o mesmo conjunto de dados. O Spark é implementado usando a linguagem de programação Scala. Scala permite que conjuntos de dados distribuídos sejam processados em um método que é o mesmo do processamento de dados locais. Além da análise de dados interativa, o Spark suporta a mineração de dados interativa. O Spark adota a computação na memória, o que facilita a computação iterativa. Por coincidência, a computação iterativa dos mesmos dados é um problema geral enfrentado pela mineração de dados. Além disso, o Spark pode ser executado em clusters do Yarn onde o Hadoop 2.0 está instalado. A razão pela qual o Spark não pode apenas manter vários recursos como tolerância a falhas de MapReduce, localização de dados e escalabilidade, mas também garantir alto desempenho e evitar I/Os de disco ocupado é que uma estrutura de abstração de memória chamada Conjunto de Dados Distribuído Resiliente (RDD) é criada para o Spark.

Abstração de memória distribuída original, por exemplo, armazenamento de valores-chave e bancos de dados, suporta atualização de pequena granularidade de status variável. Isso requer backup de dados ou atualizações de log para garantir a tolerância a falhas. Consequentemente,

uma grande quantidade de consumo de I/O é trazida para fluxos de trabalho intensivos de dados. Para o RDD, ele tem apenas um conjunto de APIs restritas e suporta apenas atualizações de grande granularidade, por exemplo, map e join. Dessa forma, o Spark só precisa registrar os logs de operação de transformação gerados durante o estabelecimento de dados para garantir a tolerância a falhas sem registrar um conjunto de dados completo. Esse registro de link de transformação de dados é uma fonte para rastrear um conjunto de dados. Geralmente, aplicações paralelas aplicam o mesmo processo de computação para um grande conjunto de dados. Portanto, o limite para a atualização de grande granularidade mencionada não é grande. Como descrito nas teses de Spark, o RDD pode funcionar como várias estruturas de computação diferentes, por exemplo, modelos de programação de MapReduce e Pregel. Além disso, o Spark permite que você explicitamente faça um processo de transformação de dados ser persistente em discos rígidos. A localização de dados é implementada permitindo que você controle partições de dados com base no valor da chave de cada registro. (Uma vantagem óbvia deste método é que duas cópias de dados a serem associados serão hash no mesmo modo.) Se o uso de memória exceder o limite físico, o Spark gravará partições relativamente grandes em discos rígidos, garantindo assim a escalabilidade.

O Spark possui os seguintes recursos:

- Rápido: a velocidade de processamento de dados do Spark é de 10 a 100 vezes maior que a do MapReduce.
- Facilidade de uso: Java, Scala e Python podem ser usados para compilar aplicações paralelas de forma simples e rápida para processar grandes quantidades de dados. O Spark fornece mais de 80 operadores para ajudá-lo a compilar aplicações paralelas.
- Universal: o Spark fornece muitas ferramentas, por exemplo, **Spark SQL** e **Spark Streaming**. Essas ferramentas podem ser combinadas de forma flexível em uma aplicação.
- Integração com Hadoop: o Spark pode ser executado diretamente em um cluster do Hadoop e ler dados existentes do Hadoop.

O componente Spark do MRS tem as seguintes vantagens:

- O componente Spark Streaming do MRS suporta o processamento de dados em tempo real, em vez de disparar conforme programado.
- O componente Spark do MRS fornece streaming estruturado e permite que você crie aplicações de streaming usando a API do conjunto de dados. O Spark suporta semântica de exatamente uma vez e junções internas e externas para fluxos.
- O componente Spark do MRS usa **pandas_udf** para substituir as funções definidas pelo usuário (UDFs) originais no PySpark para processar dados, o que reduz a duração do processamento de 60% a 90% (afetado por operações específicas).
- O componente Spark do MRS também suporta processamento de dados de grafos e permite a modelagem usando grafos durante a computação de grafos.
- O Spark SQL do MRS é compatível com alguma sintaxe do Hive (com base nas 64 instruções SQL do conjunto de testes Hive-Test-benchmark) e sintaxe SQL padrão (com base nas 99 instruções SQL do conjunto de testes TPC-DS).

Para obter detalhes sobre a arquitetura e os princípios do Spark, visite **https://spark.apache.org/docs/3.1.1/quick-start.html**.

Arquitetura

Figura 6-84 descreve a arquitetura do Spark e **Tabela 6-22** lista os módulos do Spark.

Figura 6-84 Arquitetura do Spark

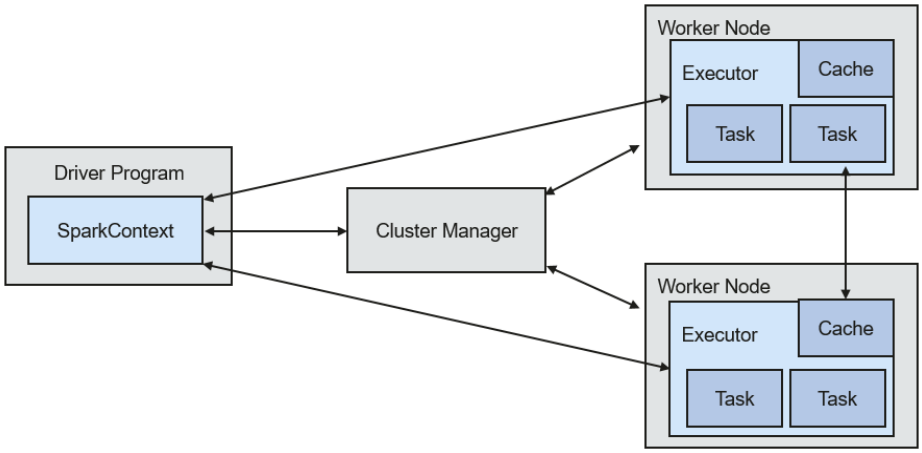


Tabela 6-22 Conceitos básicos

Módulo	Descrição
Gerenciador de cluster	O gerenciador de cluster gerencia os recursos no cluster. O Spark oferece suporte a vários gerenciadores de cluster, incluindo Mesos, Yarn e o gerenciador de cluster Standalone fornecido com o Spark.
Aplicação	Aplicação Spark. Consiste em um programa de driver e vários executores.
Modo de implementação	Implementação no modo de cluster ou de cliente. No modo de cluster, o driver é executado em um nó dentro do cluster. No modo de cliente, o driver é executado no cliente (fora do cluster).
Programa do driver	O processo principal da aplicação Spark. Ele executa a função **main()** de uma aplicação e cria SparkContext. Ele é usado para analisar aplicações, gerar estágios e agendar tarefas para executores. Normalmente, o SparkContext representa o Programa de drivers.
Executor	Um processo iniciado em um nó de trabalho. Ele é usado para executar tarefas e gerenciar e processar os dados usados em aplicações. Uma aplicação Spark geralmente contém vários executores. Cada executor recebe comandos do driver e executa uma ou várias tarefas.
Nó de trabalho	Um nó que inicia e gerencia executores e recursos em um cluster.
Job	Um job consiste em várias tarefas simultâneas. Um operador de ação (por exemplo, um operador de coleta) mapeia para um job.
Estágio	Cada job consiste em vários estágios. Cada estágio é um conjunto de tarefas, que é separado por Grafo Direcionado Acíclico (DAG).
Tarefa	Uma tarefa carrega a unidade de computação das lógicas de serviço. É a unidade de trabalho mínima que pode ser executada na plataforma Spark. Uma aplicação pode ser dividida em várias tarefas com base no plano de execução e na quantidade de computação.

Princípio de execução da aplicação Spark

Figura 6-85 mostra a arquitetura da aplicação Spark em execução. O processo de execução é o seguinte:

1. Uma aplicação está sendo executada no cluster como uma coleção de processos. Driver coordena a execução da aplicação.
2. Para executar uma aplicação, o Driver se conecta ao gerenciador de cluster (como Standalone, Mesos e Yarn) para solicitar os recursos do executor e iniciar o ExecutorBackend. O gerenciador de cluster agenda recursos entre diferentes aplicações. O Driver agenda DAGs, divide estágios e gera tarefas para a aplicação ao mesmo tempo.
3. Em seguida, o Spark envia os códigos da aplicação (os códigos transferidos para **SparkContext** que são definidos por JAR ou Python) a um executor.
4. Depois que todas as tarefas forem concluídas, a execução da aplicação do usuário é interrompida.

Figura 6-85 Arquitetura de execução da aplicação Spark

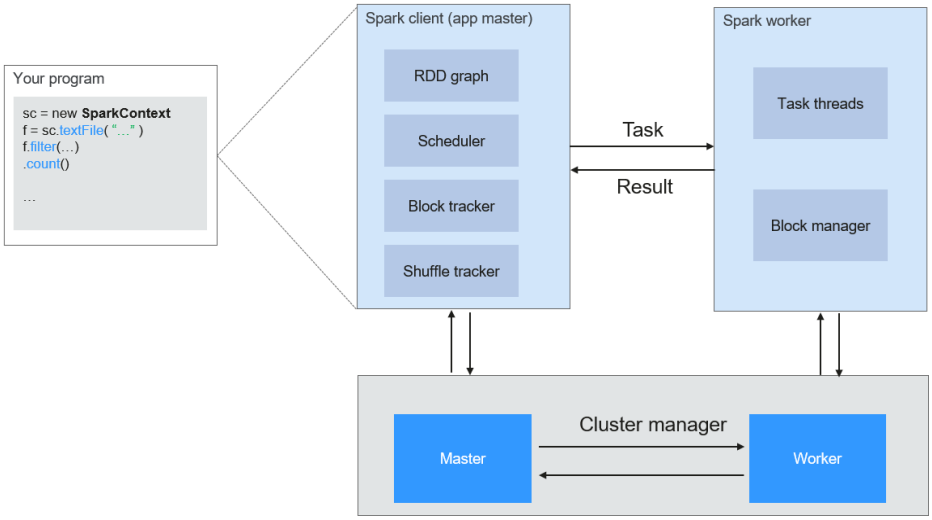
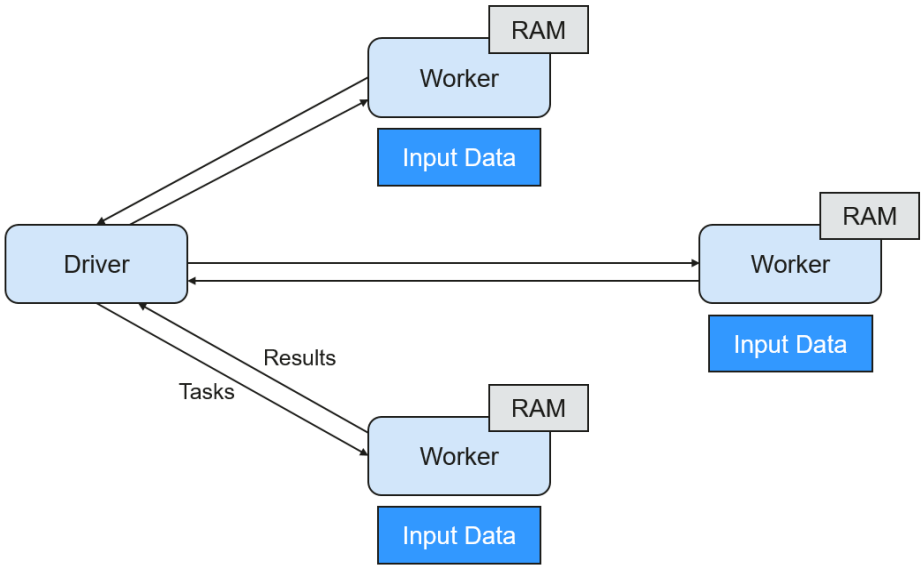


Figura 6-86 mostra os modos Master e Worker adotados pelo Spark. Um usuário envia uma aplicação no cliente Spark e, em seguida, o agendador divide um job em várias tarefas e envia as tarefas a cada Worker para execução. Cada Worker relata os resultados do cálculo para o Driver (Master) e, em seguida, o Driver agrega e retorna os resultados para o cliente.

Figura 6-86 Modo Master-Worker do Spark



Observe o seguinte sobre a arquitetura:

- As aplicações são isoladas umas das outras.

  Cada aplicação tem um processo de executor independente, e cada executor inicia vários threads para executar tarefas em paralelo. Seja em termos de agendamento ou execução de tarefas em executores. Cada motorista agenda suas próprias tarefas de forma independente. Diferentes tarefas de aplicações são executadas em diferentes JVMs, ou seja, diferentes executores.
- Diferentes aplicações Spark não compartilham dados, a menos que os dados sejam armazenados no sistema de armazenamento externo, como o HDFS.
- É aconselhável implementar o programa Driver em um local próximo ao nó de trabalho porque o programa Driver agenda tarefas no cluster. Por exemplo, implemente o programa Driver na rede onde o nó de trabalho está localizado.

O Spark no YARN pode ser implementado em dois modos:

- No modo Yarn-cluster, o driver do Spark é executado dentro de um processo ApplicationMaster que é gerenciado pelo Yarn no cluster. Depois que o ApplicationMaster é iniciado, o cliente pode sair sem interromper a execução do serviço.
- No modo Yarn-client, o driver é iniciado no processo de cliente e o processo ApplicationMaster é usado apenas para aplicar recursos do cluster de fio.

Princípio do Spark Streaming

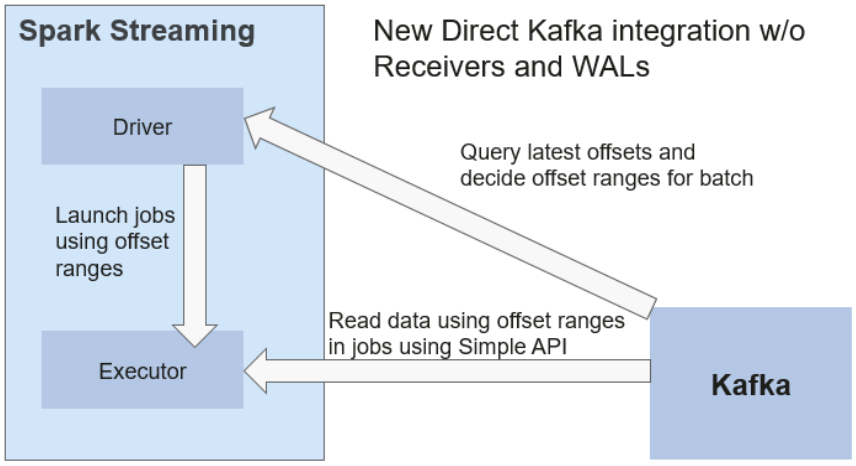
O Spark Streaming é uma estrutura de computação em tempo real construída sobre o Spark, que expande a capacidade de processar dados de streaming em massa. Atualmente, o Spark suporta os seguintes métodos de processamento de dados:

- Streaming direto

  Na abordagem de streaming direto, a API direta é usada para processar dados. Tomemos como exemplo a API direta do Kafka. A API direta fornece o local de deslocamento do qual cada intervalo de lote será lido, o que é muito mais simples do que iniciar um receptor para receber continuamente dados do Kafka e dados gravados em logs de

gravação antecipada (WALs). Em seguida, cada job em lote está em execução e os dados de deslocamento correspondentes estão prontos no Kafka. Essas informações de deslocamento podem ser armazenadas com segurança no arquivo de ponto de verificação e lidas por aplicações que falharam ao iniciar.

Figura 6-87 Transmissão de dados através da API Direct Kafka



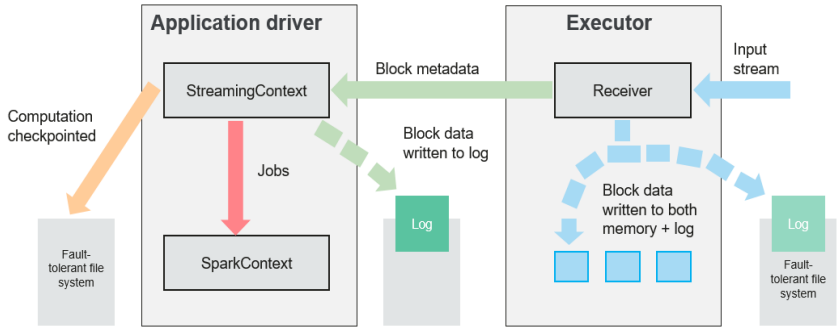
Após a falha, o Spark Streaming pode ler dados do Kafka novamente e processar o segmento de dados. O resultado do processamento é o mesmo, não importa se o Spark Streaming falha ou não, porque a semântica é processada apenas uma vez.

A API direta não precisa usar o WAL e os receptores e garante que cada registro do Kafka seja recebido apenas uma vez, o que é mais eficiente. Desta forma, o Spark Streaming e o Kafka podem ser bem integrados, fazendo com que os canais de streaming sejam apresentados com alta tolerância a falhas, alta eficiência e facilidade de uso. Portanto, é aconselhável usar o Streaming direto para processar dados.

- Receptor

  Quando uma aplicação Spark Streaming é iniciada, o StreamingContext relacionado (a base de todas as funções de streaming) usa o SparkContext para iniciar o receptor e se tornar uma tarefa de longa duração. Esses receptores recebem e salvam dados de streaming na memória do Spark para processamento. **Figura 6-88** mostra o ciclo de vida da transferência de dados.

Figura 6-88 Ciclo de vida da transferência de dados



a. Receber dados (seta azul).

O receptor divide um fluxo de dados em uma série de blocos e os armazena na memória do executor. Além disso, depois que o WAL é habilitado, ele grava dados no WAL do sistema de arquivos tolerante a falhas.

b. Notificar o driver (seta verde).

Os metadados no bloco recebido são enviados para StreamingContext no driver. Os metadados incluem:

- ID de referência de bloco usado para localizar a posição dos dados na memória do Executor.
- Informações de deslocamento de dados de bloco em logs (se a função WAL estiver ativada).

c. Dados do processo (seta vermelha).

Para cada lote de dados, o StreamingContext usa informações de bloco para gerar conjuntos de dados distribuídos resilientes (RDDs) e jobs. O StreamingContext executa jobs executando tarefas para processar blocos na memória do executor.

d. Periodicamente definir pontos de verificação (setas laranja).

Para tolerância a falhas, o StreamingContext define periodicamente pontos de verificação e os salva em sistemas de arquivos externos.

Tolerância a falhas

O Spark e seu RDD permitem o processamento contínuo de falhas de qualquer nó de trabalho no cluster. O Spark Streaming é construído em cima do Spark. Portanto, o nó de trabalho do Spark Streaming também tem a mesma capacidade de tolerância a falhas. No entanto, o Spark Streaming precisa ser executado corretamente em caso de execução de longo tempo. Portanto, o Spark deve ser capaz de se recuperar de falhas através do processo de driver (processo principal que coordena todos os Workers). Isso representa desafios para a tolerância a falhas do driver do Spark, pois o driver do Spark pode ser qualquer aplicação de usuário implementado em qualquer modo de computação. No entanto, o Spark Streaming possui uma arquitetura de computação interna. Ou seja, ele executa periodicamente o mesmo cálculo do Spark em cada dado de lote. Tal arquitetura permite que ele armazene periodicamente pontos de verificação para espaço de armazenamento confiável e os recupere após a reinicialização do Driver.

Para dados de origem, como arquivos, o mecanismo de recuperação de Driver pode garantir zero perda de dados, pois todos os dados são armazenados em um sistema de arquivos tolerante a falhas, como o HDFS. No entanto, para outras fontes de dados, como Kafka e Flume, alguns dados recebidos são armazenados em cache apenas na memória e podem ser perdidos antes de serem processados. Isso é causado pelo modo de operação de distribuição das aplicações Spark. Quando o processo de driver falha, todos os executores em execução no Gerenciador de clusters, juntamente com todos os dados na memória, são encerrados. Para evitar essa perda de dados, a função WAL é adicionada ao Spark Streaming.

O WAL é frequentemente usado em bancos de dados e sistemas de arquivos para garantir a persistência de qualquer operação de dados. Ou seja, primeiro registre uma operação em um log persistente e execute essa operação nos dados. Se a operação falhar, o sistema é recuperado lendo o registro e reaplicando a operação predefinida. A seguir, descrevemos como usar o WAL para garantir a persistência dos dados recebidos:

Receptor é usado para receber dados de fontes de dados como Kafka. Como uma tarefa de longa duração no Executor, o Receptor recebe dados e também confirma os dados recebidos, se suportados por fontes de dados. Os dados recebidos são armazenados na memória do Executor e o Driver entrega uma tarefa ao Executor para processamento.

Depois que o WAL é habilitado, todos os dados recebidos são armazenados em arquivos de log no sistema de arquivos tolerante a falhas. Portanto, os dados recebidos não perdem mesmo se o Spark Streaming falhar. Além disso, o receptor verifica a exatidão dos dados recebidos somente depois que os dados são pré-gravados em logs. Os dados armazenados em cache, mas não armazenados, podem ser enviados novamente por fontes de dados após a reinicialização do driver. Esses dois mecanismos garantem perda zero de dados. Ou seja, todos os dados são recuperados de logs ou reenviados por fontes de dados.

Para ativar a função WAL, execute as seguintes operações:

- Defina **streamingContext.checkpoint** para configurar o diretório de ponto de verificação, que é um caminho de arquivo do HDFS usado para armazenar pontos de verificação de streaming e WALs.
- Defina **spark.streaming.receiver.writeAheadLog.enable** do SparkConf como **true** (o valor padrão é **false**).

Depois que o WAL é ativado, todos os receptores têm a vantagem de se recuperar de dados recebidos confiáveis. É aconselhável desabilitar o mecanismo de várias réplicas porque o sistema de arquivos tolerante a falhas do WAL também pode replicar os dados.

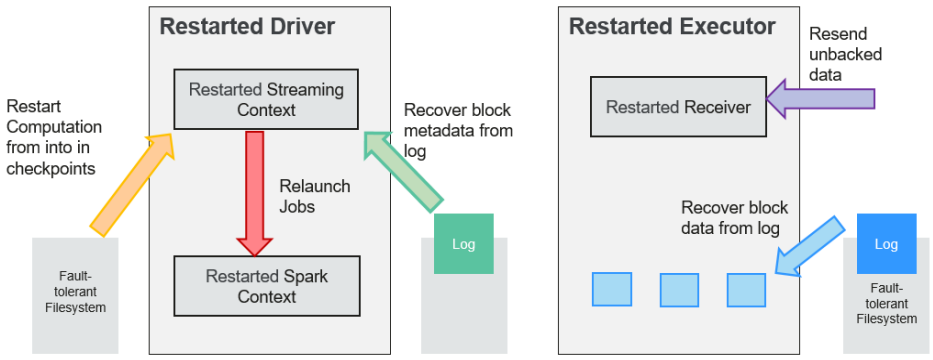
NOTA

A taxa de transferência de recebimento de dados é reduzida depois que o WAL é habilitado. Todos os dados são gravados no sistema de arquivos tolerante a falhas. Como resultado, a taxa de transferência de gravação do sistema de arquivos e a largura de banda da rede para replicação de dados podem se tornar o gargalo potencial. Para resolver esse problema, é aconselhável criar mais receptores para aumentar o grau de paralelismo de recebimento de dados ou usar um hardware melhor para melhorar a taxa de transferência do sistema de arquivos tolerante a falhas.

Processo de recuperação

Quando um driver com falha for reiniciado, reinicie-o da seguinte maneira:

Figura 6-89 Processo de recuperação de computação



1. Recupere a computação. (Seta laranja)

   Use as informações do ponto de verificação para reiniciar o Driver, reconstruir o SparkContext e reiniciar o Receptor.

2. Recupere bloco de metadados. (Seta verde)

   Essa operação garante que todos os blocos de metadados necessários sejam recuperados para continuar a recuperação computacional subsequente.

3. Relance jobs inacabados. (Seta vermelha)

Os metadados recuperados são usados para gerar RDDs e jobs correspondentes para processamento em lote interrompido devido a falhas.

4. Leia dados de bloco salvos em logs. (Seta azul)

   Os dados de bloco são lidos diretamente dos WALs durante a execução dos jobs anteriores e, portanto, todos os dados essenciais armazenados de forma confiável nos logs são recuperados.

5. Reenvie dados não confirmados. (Seta roxa)

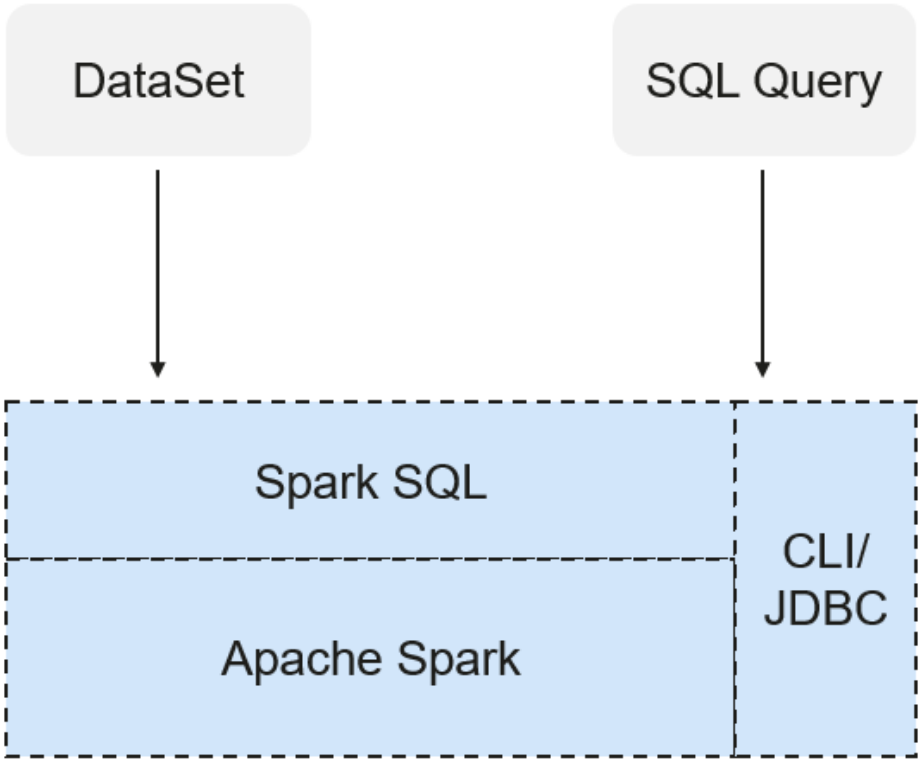
   Os dados que são armazenados em cache, mas não armazenados em logs após falhas, são reenviados por fontes de dados, porque o receptor não confirma os dados.

Portanto, usando WALs e um Receptor confiável, o Spark Streaming pode evitar a perda de dados de entrada causada por falhas de Driver.

Princípio do SparkSQL e do DataSet

SparkSQL

Figura 6-90 SparkSQL e DataSet



Spark SQL é um módulo para processamento de dados estruturados. Na aplicação Spark, instruções SQL ou APIs de DataSet podem ser usadas para consultar dados estruturados.

Spark SQL e DataSet também fornecem um método universal para acessar várias fontes de dados, como Hive, CSV, Parquet, ORC, JSON e JDBC. Essas fontes de dados também permitem a interação de dados. O Spark SQL reutiliza a lógica de processamento de front-end do Hive e o módulo de processamento de metadados. Com o Spark SQL, você pode consultar diretamente os dados existentes do Hive.

Além disso, o Spark SQL também fornece APIs, CLI e APIs de JDBC, permitindo diversos acessos ao cliente.

DDL/DML de Spark SQL nativo

No Spark 1.5, muitos comandos de Linguagem de Definição de Dados (DDL)/Linguagem de Manipulação de Dados (DML) são empurrados para baixo e executados no Hive, causando acoplamento com o Hive e inflexibilidade, como relatórios e resultados de erros inesperados.

O Spark 3.1.1 realiza a localização de comandos e substitui o Hive por DDL/DML do Spark SQL Nativo para executar comandos DDL/DML. Além disso, o desacoplamento do Hive é realizado e os comandos podem ser personalizados.

DataSet

Um DataSet é uma coleção fortemente tipada de objetos específicos de domínio que podem ser transformados em paralelo usando operações funcionais ou relacionais. Cada Dataset também tem uma exibição não tipada chamada DataFrame que é um Dataset de linha.

DataFrame é um conjunto de dados estruturado e distribuído que consiste em várias colunas. DataFrame é igual a uma tabela no banco de dados de relacionamento ou o DataFrame no R/Python. DataFrame é o conceito mais básico do Spark SQL, que pode ser criado usando vários métodos, como o conjunto de dados estruturado, a tabela do Hive, o banco de dados externo ou o RDD.

As operações disponíveis no DataSets são divididas em transformações e ações.

- Uma operação de transformação pode gerar um novo DataSet,
  por exemplo, **map**, **filter**, **select** e **aggregate (groupBy)**.
- Uma operação de ação pode acionar a computação e retornar resultados,
  por exemplo, **count**, **show** ou gravar dados no sistema de arquivos.

Você pode usar um dos seguintes métodos para criar um DataSet:

- A maneira mais comum é apontar o Spark para alguns arquivos em sistemas de armazenamento, usando a função **read** disponível em um SparkSession.
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people =
  spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- Você também pode criar um DataSet usando a operação de transformação disponível em um existente.
  Por exemplo, aplique a operação de mapa em um DataSet existente para criar um DataSet:
  ```
  val names = people.map(_.name) // In Scala: names is Dataset.
  Dataset<String> names = people.map((Person p) -> p.name,
  Encoders.STRING)); // Java
  ```

CLI e JDBCServer

Além das APIs de programação, o Spark SQL também fornece as APIs de CLI/JDBC.

- Os scripts **spark-shell** e **spark-sql** podem fornecer a CLI para depuração.
- JDBCServer fornece APIs de JDBC. Sistemas externos podem enviar diretamente solicitações JDBC para calcular e analisar dados estruturados.

Princípio de SparkSession

SparkSession é uma API unificada para programação Spark e pode ser considerada como uma entrada unificada para leitura de dados. A SparkSession fornece um único ponto de entrada

para executar muitas operações anteriormente espalhadas por várias classes e também fornece métodos de acesso a essas classes antigas para maximizar a compatibilidade.

Uma SparkSession pode ser criada usando um padrão de construtor. O construtor reutilizará automaticamente a SparkSession existente se houver uma SparkSession ou criará uma SparkSession se ela não existir. Durante as transações de I/O, as definições do item de configuração no construtor são automaticamente sincronizadas com o Spark e o 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()
```

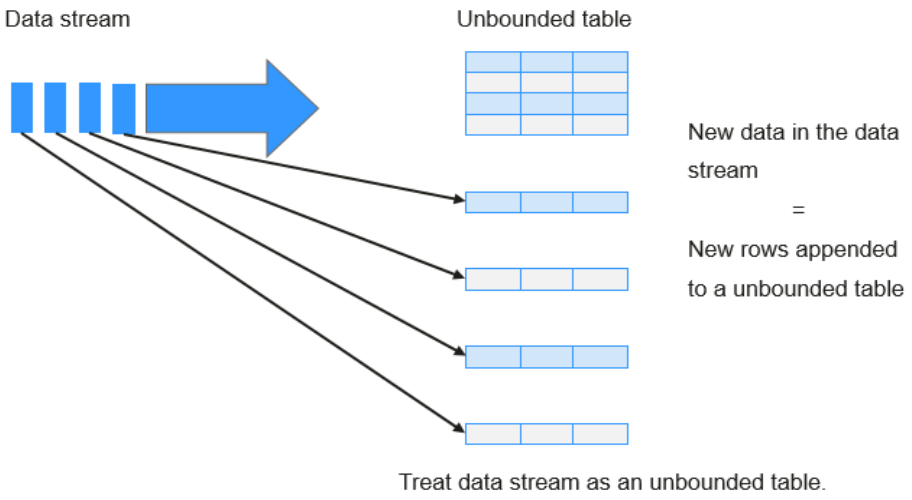
- SparkSession pode ser usada para executar consultas SQL em dados e retornar resultados como DataFrame.
  ```
  sparkSession.sql("select * from person").show
  ```
- A SparkSession pode ser usada para definir itens de configuração durante a execução. Esses itens de configuração podem ser substituídos por variáveis em instruções SQL.
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- A SparkSession também inclui um método "catalog" que contém métodos para trabalhar com o Metastore (catálogo de dados). Depois que esse método é usado, um conjunto de dados é retornado, que pode ser executado usando a mesma API de Dataset.
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```
- O SparkContext subjacente pode ser acessado pela API de SparkContext da SparkSession.
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Princípio do Streaming Estruturado

O Streaming Estruturado é um mecanismo de processamento de fluxo construído no motor Spark SQL. Você pode usar a API de Dataset/DataFrame em Scala, Java, Python ou R para expressar agregações de streaming, janelas de tempo de evento e junções fluxo-fluxo. Se os dados de streaming forem produzidos de forma incremental e contínua, o Spark SQL continuará processando os dados e sincronizando o resultado com o conjunto de resultados. Além disso, o sistema garante de ponta a ponta garantias de tolerância a falhas exatas através de pontos de verificação e WALs.

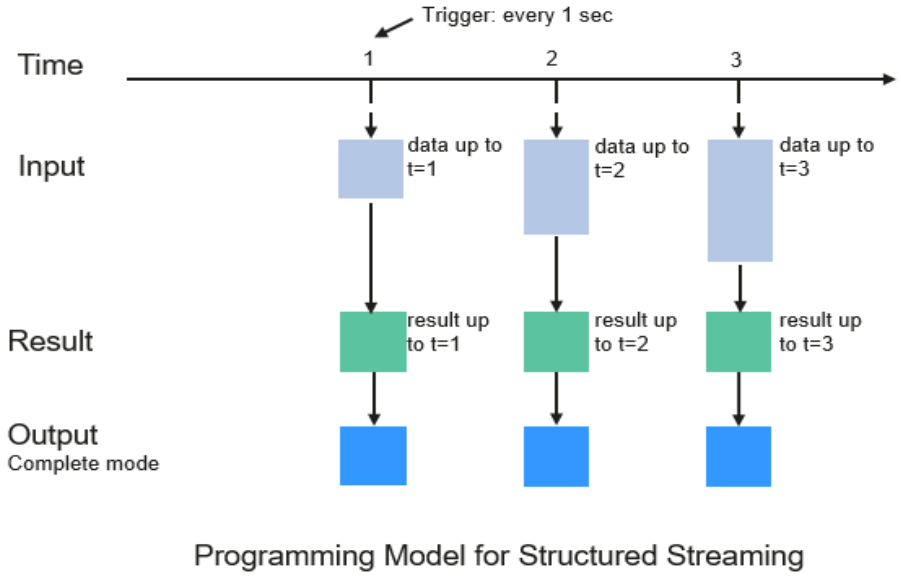
O núcleo do Streaming Estruturado é tomar dados de streaming como uma tabela de banco de dados incremental. Semelhante ao modelo de processamento de blocos de dados, o modelo de processamento de dados de streaming aplica operações de consulta em uma tabela de banco de dados estática à computação de streaming, e o Spark usa instruções SQL padrão para consulta, para obter dados da tabela incremental e ilimitada.

Figura 6-91 Tabela ilimitada de Streaming Estruturado



Cada operação de consulta irá gerar uma tabela de resultados. Em cada intervalo de gatilho, os dados atualizados serão sincronizados com a tabela de resultados. Sempre que a tabela de resultados for atualizada, o resultado atualizado será gravado em um sistema de armazenamento externo.

Figura 6-92 Modelo de processamento de dados de Streaming Estruturado



Os modos de armazenamento de Streaming Estruturado na fase de saída são os seguintes:

- Modo completo: os conjuntos de resultados atualizados são gravados no sistema de armazenamento externo. A operação de gravação é realizada por um conector do sistema de armazenamento externo.
- Modo de adição: se um intervalo for acionado, apenas os dados adicionados na tabela de resultados serão gravados em um sistema externo. Isso é aplicável somente nas consultas em que não se espera que as linhas existentes na tabela de resultados sejam alteradas.

- Modo de atualização: Se um intervalo for acionado, somente os dados atualizados na tabela de resultados serão gravados em um sistema externo, que é a diferença entre o Modo completo e o Modo de atualização.

Conceitos básicos

- **RDD**

  Conjunto de dados distribuídos resilientes (RDD) é um conceito central do Spark. Indica um conjunto de dados distribuído somente leitura e particionado. Parciais ou todos os dados deste conjunto de dados podem ser armazenados em cache na memória e reutilizados entre cálculos.

  Criação de RDD

  - Um RDD pode ser criado a partir da entrada do HDFS ou de outros sistemas de armazenamento compatíveis com o Hadoop.
  - Um novo RDD pode ser convertido a partir de um RDD pai.
  - Um RDD pode ser convertido a partir de uma coleção de conjuntos de dados através da codificação.

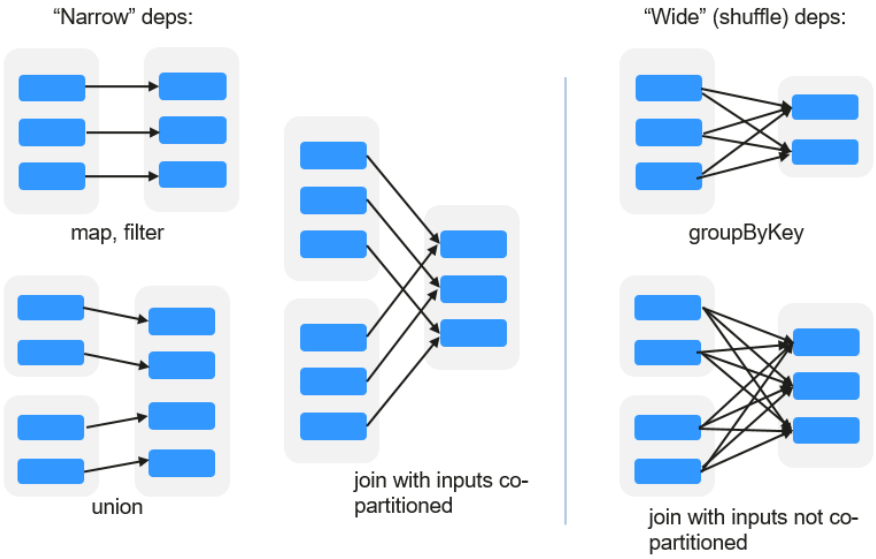
  Armazenamento de RDD

  - Você pode selecionar diferentes níveis de armazenamento para armazenar um RDD para reutilização. (Há 11 níveis de armazenamento para armazenar um RDD.)
  - Por padrão, o RDD é armazenado na memória. Quando a memória é insuficiente, o RDD transborda para o disco.

- **Dependência de RDD**

  A dependência de RDD inclui a dependência estreita e a dependência ampla.

  Figura 6-93 Dependência de RDD

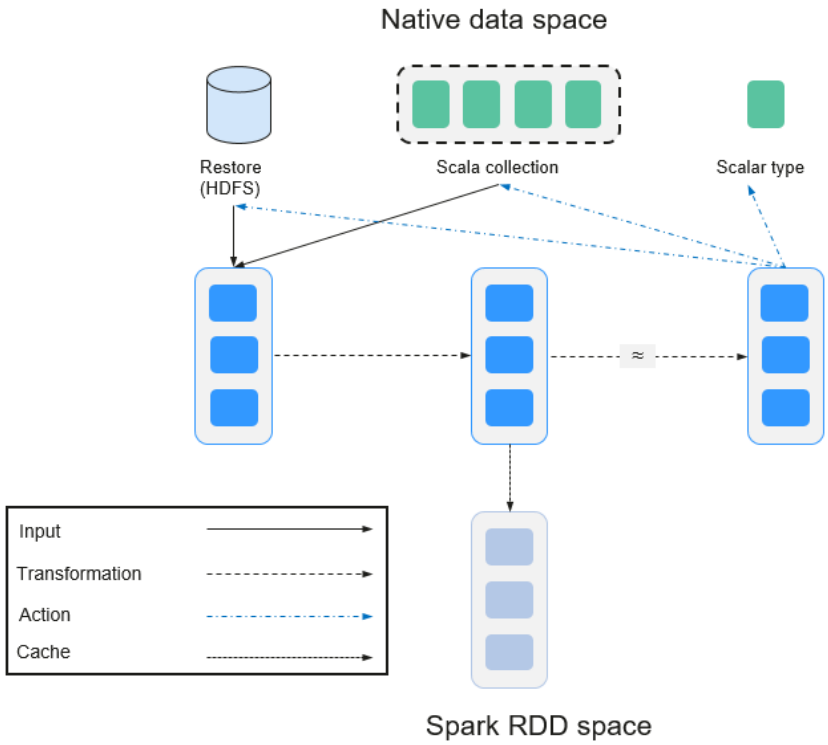
  - **Dependência estreita**: cada partição do RDD pai é usada por no máximo uma partição do RDD filho.
  - **Dependência ampla**: as partições do RDD filho dependem de todas as partições do RDD pai.

A dependência estreita facilita a otimização. Logicamente, cada operador do RDD é um fork/join (join não é o operador de junção mencionado acima, mas a barreira usada para sincronizar várias tarefas simultâneas); fork o RDD para cada partição e, em seguida, execute a computação. Após a computação, junte os resultados e, em seguida, execute a operação fork/join no próximo operador do RDD. É antieconômico traduzir diretamente o RDD em implementação física. A primeira é que cada RDD (mesmo resultado intermediário) precisa ser fisicalizado em memória ou armazenamento, o que é demorado e ocupa muito espaço. A segunda é que, como uma barreira global, a operação join é muito cara e todo o processo de join será retardado pelo nó mais lento. Se as partições do RDD filho dependerem estritamente das do RDD pai, os dois processos de fork/join podem ser combinados para implementar a otimização de fusão clássica. Se a relação na sequência contínua do operador for de dependência estreita, vários processos de fork/join podem ser combinados para reduzir um grande número de barreiras globais e eliminar a fisicalização de muitos resultados intermediários do RDD, o que melhora muito o desempenho. Isso é chamado de otimização de pipeline no Spark.

- **Transformação e ação (Operações do RDD)**

  As operações no RDD incluem transformação (o valor de retorno é um RDD) e ação (o valor de retorno não é um RDD). **Figura 6-94** mostra o processo de operação do RDD. A transformação é lenta, o que indica que a transformação de um RDD para outro RDD não é executada imediatamente. O Spark registra apenas a transformação, mas não a executa imediatamente. A computação real é iniciada somente quando a ação é iniciada. A ação retorna resultados ou grava os dados do RDD no sistema de armazenamento. A ação é a força motriz para o Spark iniciar a computação.

Figura 6-94 Operação de RDD



Os dados e o modelo de operação do RDD são bem diferentes dos do Scala.

```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
```



```
errors.cache()
errors.count()
```

a. O operador de textFile lê arquivos de log do HDFS e retorna arquivos (como um RDD).

b. O operador de filtro filtra as linhas com **ERROR** e as atribui a erros (um novo RDD). O operador de filtro é uma transformação.

c. O operador de cache armazena em caches os erros para uso futuro.

d. O operador de contagem retorna o número de linhas de erros. O operador de contagem é uma ação.

A transformação inclui os seguintes tipos:

- Os elementos RDD são considerados elementos simples.

  A entrada e a saída têm a relação um-para-um, e a estrutura de partição do RDD resultante permanece inalterada, por exemplo, map.

  A entrada e a saída têm a relação um-para-muitos, e a estrutura de partição do RDD resultante permanece inalterada, por exemplo, flatMap (um elemento torna-se uma sequência contendo vários elementos após map e, em seguida, achata para vários elementos).

  A entrada e a saída têm a relação um-para-um, mas a estrutura da partição do RDD resultante muda, por exemplo, union (dois RDDs integram-se a um RDD, e o número de partições torna-se a soma do número de partições de dois RDDs) e coalesce (as partições são reduzidas).

  Operadores de alguns elementos são selecionados a partir da entrada, como filter, distinct (elementos duplicados são excluídos), subtract (elementos que só existem neste RDD são mantidos) e sample (amostras são tomadas).

- Os elementos do RDD são considerados pares chave-valor.

  Executar o cálculo um-para-um no RDD único, como mapValues (o modo de partição do RDD de origem é mantido, o que é diferente do mapa).

  Classificar o RDD único, como sort e partitionBy (particionamento com consistência, o que é importante para a otimização local).

  Reestruturar e reduzir o RDD único com base na chave, como groupByKey e reduceByKey.

  Juntar-se e reestruturar dois RDDs com base na chave, como join e cogroup.

  > NOTA
  >
  > As três operações posteriores que envolvem ordenação são chamadas de operações de shuffle.

A ação inclui os seguintes tipos:

- Gerar itens de configuração de escalar, como **count** (o número de elementos no RDD retornado), **reduce**, **fold/aggregate** (o número de itens de configuração de escalar que são retornados) e **take** (o número de elementos antes do retorno).
- Gerar a coleção Scala, como **collect** (importar todos os elementos do RDD para a coleção Scala) e **lookup** (procurar todos os valores correspondentes à chave).
- Gravar dados no armazenamento, como **saveAsTextFile** (que corresponde ao **textFile** anterior).
- Pontos de verificação, como o operador do **checkpoint**. Quando o Lineage é bastante longo (o que ocorre frequentemente na computação gráfica), leva um longo período de tempo para executar toda a sequência novamente quando ocorre uma

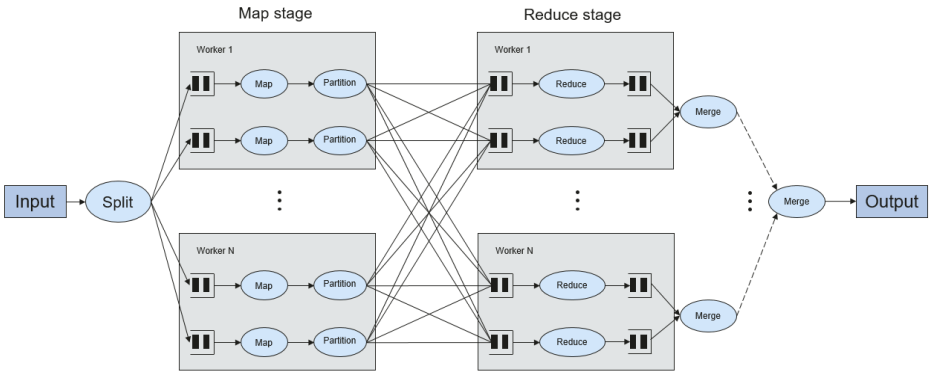
falha. Nesse caso, o ponto de verificação é usado como o ponto de verificação para gravar os dados atuais no armazenamento estável.

- **Shuffle**

  Shuffle é uma fase específica na estrutura MapReduce, que está localizada entre a fase Map e a fase Reduce. Se os resultados de saída de Map forem usados pela Reduce, os resultados de saída deverão ser hash com base em uma chave e distribuídos para cada Redutor. Esse processo é chamado de Shuffle. Shuffle envolve a leitura e a escrita do disco e a transmissão da rede, de modo que o desempenho de Shuffle afete diretamente a eficiência da operação de todo o programa.

  A figura abaixo mostra todo o processo do algoritmo MapReduce.

  Figura 6-95 Processo de algoritmo

  Shuffle é uma ponte para conectar dados. A seguir, descreve-se a implementação do shuffle no Spark.

  Shuffle divide um job do Spark em vários estágios. Os estágios anteriores contêm uma ou mais ShuffleMapTasks, e o último estágio contém uma ou mais ResultTasks.

- **Estrutura da aplicação Spark**

  A estrutura da aplicação Spark inclui o SparkContext inicializado e o programa principal.

  - SparkContext inicializado: constrói o ambiente operacional da aplicação Spark.

    Constrói o objeto SparkContext. O seguinte é um exemplo:

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    Descrição do parâmetro:

    master: indica a cadeia de ligação. Os modos de link incluem local, Yarn-cluster e Yarn-client.

    appName: indica o nome da aplicação.

    SparkHome: indica o diretório em que o Spark está instalado no cluster.

    jars: indica o pacote de código e dependência de uma aplicação.

  - Programa principal: processa dados.

  Para obter detalhes sobre como enviar uma inscrição, visite **https://spark.apache.org/docs/3.1.1/submitting-applications.html**.

- **Comandos shell do Spark**

  Os comandos básicos shell do Spark suportam o envio de aplicações Spark. Os comandos shell do Spark são os seguintes:


```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

Descrição do parâmetro:

--class: indica o nome da classe de uma aplicação Spark.

--master: indica o mestre ao qual a aplicação Spark se vincula, como Yarn-client e Yarn-cluster.

application-jar: indica o caminho do arquivo JAR da aplicação Spark.

application-arguments: indica o parâmetro necessário para enviar a aplicação Spark. Este parâmetro pode ser deixado em branco.

- **Servidor JobHistory do Spark**

  A IU da Web do Spark é usada para monitorar os detalhes em cada fase da estrutura do Spark de um job do Spark em execução ou histórico e fornecer a exibição do log, o que ajuda os usuários a desenvolver, configurar e otimizar o job em unidades mais refinadas.

6.27.2 Solução HA do Spark

Princípios e solução de implementação de HA de instância multi-ativa de Spark

Com base no JDBCServer existente na comunidade, o modo de instância multi-ativa é usado para obter HA. Neste modo, vários JDBCServers coexistem no cluster e o cliente pode conectar aleatoriamente qualquer JDBCServer para executar operações de serviço. Quando um ou vários JDBCServers param de funcionar, um cliente pode se conectar a outro JDBCServer normal.

Comparado com o modo HA ativo/em espera, o modo de instância multi-ativa tem as seguintes vantagens:

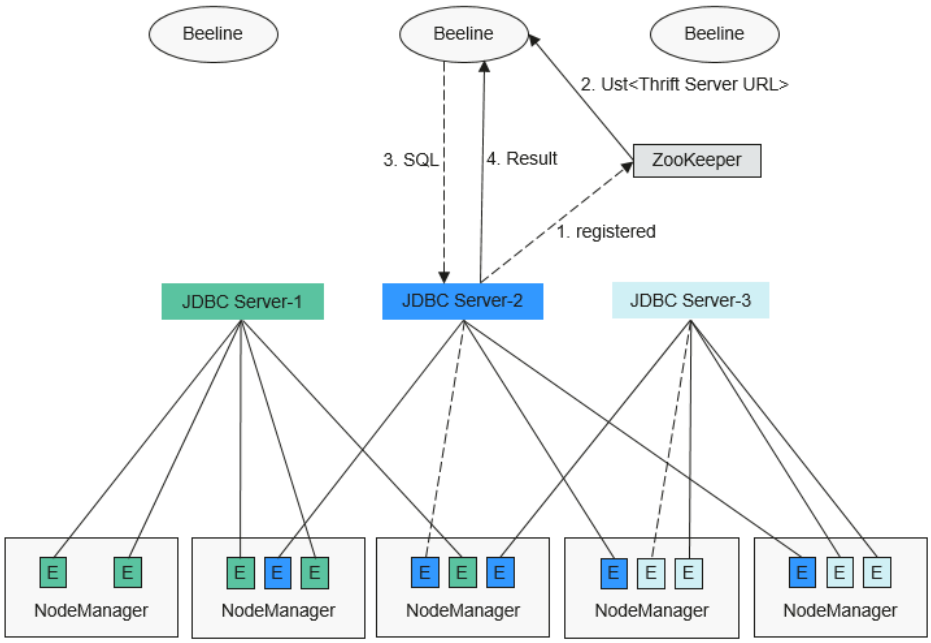
- No HA ativo/em espera, quando a alternância ativa/em espera ocorre, o período indisponível não pode ser controlado pelo JDBCServer, mas depende dos recursos do serviço de Yarn.
- No Spark, o Thrift JDBC semelhante ao HiveServer2 fornece serviços e usuários acessam serviços por meio da Beeline e da API de JDBC. Portanto, a capacidade de processamento do cluster de JDBCServer depende da capacidade de ponto único do servidor primário, e a escalabilidade é insuficiente.

O modo HA de instância multi-ativa não só pode impedir a interrupção do serviço causada pela alternância, mas também permite a expansão do cluster para melhorar a alta simultaneidade.

- **Implementação**

  A figura a seguir mostra o princípio básico do HA da instância multi-ativa do Spark JDBCServer.

Figura 6-96 Spark JDBCServer HA



1. Quando um JDBCServer é iniciado, ele se registra no ZooKeeper escrevendo as informações do nó em um diretório especificado. As informações do nó incluem o endereço IP da instância, o número da porta, a versão e o número de série.
2. Para se conectar ao JDBCServer, o cliente deve especificar o namespace, que é o diretório das instâncias do JDBCServer em ZooKeeper. Durante a conexão, uma instância do JDBCServer é selecionada aleatoriamente no namespace especificado.
3. Após a conexão ser bem-sucedida, o cliente envia instruções SQL para o JDBCServer.
4. JDBCServer executa instruções SQL recebidas e retorna resultados para o cliente.

Se o HA da instância multi-ativa do Spark JDBCServer estiver ativada, todas as instâncias do JDBCServer serão independentes e equivalentes. Quando uma instância do JDBCServer é interrompida durante a atualização, outras instâncias do JDBCServer podem aceitar a solicitação de conexão do cliente.

As regras abaixo devem ser seguidas no HA da instância multi-ativa do Spark JDBCServer.

- Se uma instância do JDBCServer sair anormalmente, nenhuma outra instância assumirá as sessões e os serviços em execução na instância anormal.
- Quando o processo do JDBCServer é interrompido, os nós correspondentes são excluídos do ZooKeeper.
- O cliente seleciona aleatoriamente o servidor, o que pode resultar em alocação desigual de sessão causada por distribuição aleatória de resultados de políticas e, finalmente, resultar em desequilíbrio de carga de instâncias.
- Depois que a instância entra no modo de manutenção (em que não são aceites novas solicitações de conexão dos clientes) , os serviços em execução na instância podem falhar quando o tempo limite do descomissionamento.
- **Conexão de URL**
  - Modo de instância multi-ativa

    No modo de instância multi-ativa, o cliente lê o conteúdo do nó do ZooKeeper e se conecta ao JDBCServer. As cadeias de conexão estão listadas abaixo.

- Modo de segurança:

  Se a autenticação Kinit estiver ativada, o JDBCURL será o seguinte:

  ```
  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

  - No JDBCURL acima, **<zkNode_IP>:<zkNode_Port>** indica o URL do ZooKeeper. Use vírgulas (,) para separar vários URLs,

    exemplo: 192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181.
  - **sparkthriftserver2x** indica o diretório do ZooKeeper, onde uma instância aleatória do JDBCServer está conectada ao cliente.

  Por exemplo, quando você usa o cliente de Beeline para conectar o JDBCServer, execute o seguinte 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*>**;"**

  Se a autenticação Keytab estiver ativada, o JDBCURL será o seguinte:

  ```
  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>
  ```

  No URL acima, <*principal_name*> indica o principal do usuário Kerberos, por exemplo, **test@**<*System domain name*>; <*path_to_keytab*> indica o caminho do arquivo Keytab correspondente a <*principal_name*>, por exemplo, **/opt/auth/test/user.keytab**.

- Modo comum:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
  <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
  rkthriftserver2x;
  ```

  Por exemplo, quando utiliza o cliente de Beeline, no modo normal, para a conexão, execute o seguinte 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 instância não multi-ativa

  Neste modo, um cliente se conecta a um nó do JDBCServer especificado. Comparado com o modo de instância multi-ativa, a cadeia de conexão neste modo não contém os parâmetros **serviceDiscoveryMode** e **zooKeeperNamespace** sobre ZooKeeper.

  Por exemplo, quando você usa o cliente de Beeline, no modo de segurança, para conectar o JDBCServer no modo de instância não multi-ativa, execute o seguinte 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

- No comando acima, **<server_IP>:<server_Port>** indica o URL do nó do JDBCServer especificado.
- **CLIENT_HOME** indica o caminho do cliente.

Exceto o método de conexão, outras operações da API do JDBCServer nos dois modos são as mesmas. Spark JDBCServer é outra implementação do HiveServer2 no Hive. Para obter detalhes sobre como usar Spark JDBCServer, consulte **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

HA multi-locatário do Spark

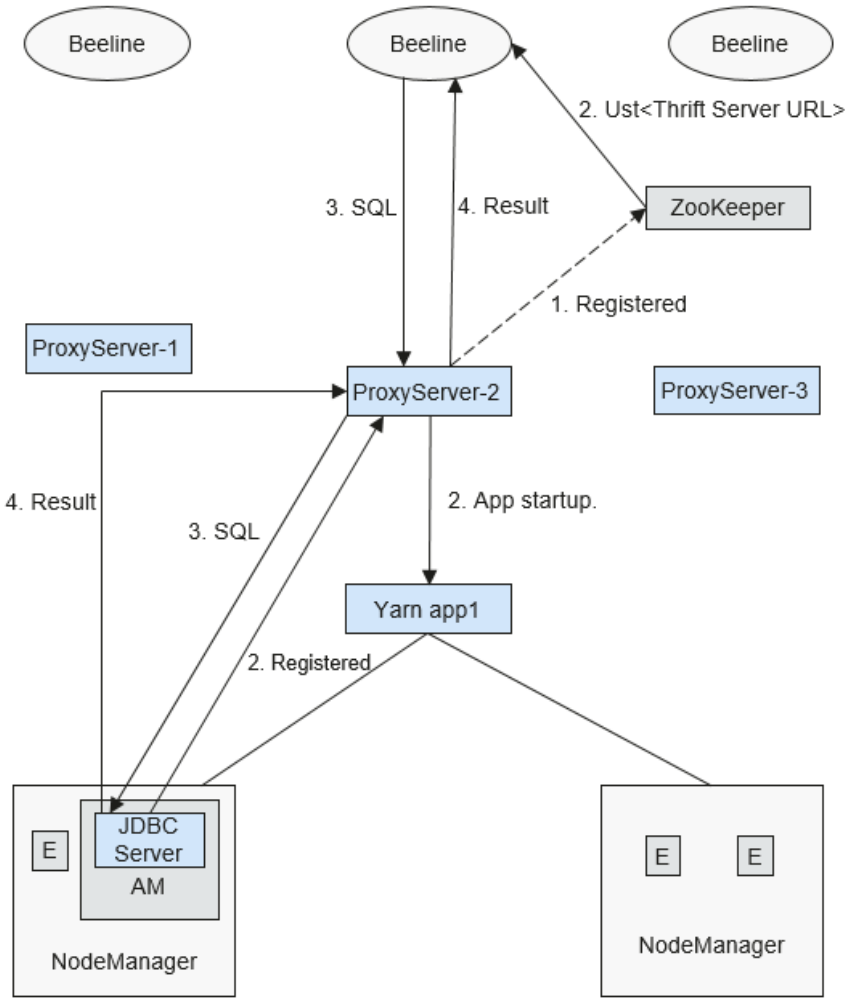
Na solução de instâncias multi-ativas do JDBCServer, o JDBCServer usa o modo Yarn-client, mas há apenas uma fila de recursos Yarn disponível. Para resolver esse problema de limitação de recursos, o modo multi-locatário é introduzido.

No modo multi-locatário, os JDBCServers são vinculados aos locatários. Cada locatário corresponde a um ou mais JDBCServers, e um JDBCServer fornece serviços para apenas um locatário. Locatários diferentes podem ser configurados com filas de Yarn diferentes para implementar o isolamento de recursos. Além disso, o JDBCServer pode ser iniciado dinamicamente conforme necessário para evitar o desperdício de recursos.

- **Implementação**

  Figura 6-97 mostra a solução HA do modo multi-locatário.

Figura 6-97 Modo multi-locatário do Spark JDBCServer



a. Quando um ProxyServer é iniciado, ele se registra no ZooKeeper escrevendo as informações do nó em um diretório especificado. As informações do nó incluem o endereço IP da instância, o número da porta, a versão e o número de série.

   NOTA

   No modo multi-locatário, a instância do JDBCServer refere-se ao ProxyServer (proxy do DBCServer).

b. Para se conectar ao ProxyServer, o cliente deve especificar um namespace, que é o diretório da instância do ProxyServer onde você deseja acessar o ZooKeeper. Quando o cliente se conecta ao ProxyServer, uma instância aleatória sob o namespace é selecionada para conexão. Para obter detalhes sobre o URL, consulte **Visão geral da conexão de URL**.

c. Depois que o cliente se conecta com sucesso ao ProxyServer que primeiro verifica se o JDBCServer de um locatário existe. Se sim, o Beeline conecta o JDBCServer. Se não, um novo JDBCServer será iniciado no modo Yarn-cluster. Após a inicialização do JDBCServer, o ProxyServer obtém o endereço IP do JDBCServer e estabelece a conexão entre o Beeline e o JDBCServer.

d. O cliente envia instruções SQL para ProxyServer que encaminha instruções para o JDBCServer conectado. JDBCServer retorna os resultados para ProxyServer que, em seguida, retorna os resultados para o cliente.

No modo HA de instância multi-ativa, todas as instâncias são independentes e equivalentes. Se uma instância for interrompida durante a atualização, outras instâncias poderão aceitar a solicitação de conexão do cliente.

- **Visão geral da conexão de URL**
  - Modo multi-locatário

    No modo multi-locatário, o cliente lê o conteúdo do nó do ZooKeeper e se conecta ao ProxyServer. As cadeias de conexão estão listadas abaixo.

    - Modo de segurança:

      Se a autenticação Kinit estiver ativada, o URL de cliente será o seguinte:

      ```
      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

      - No URL acima, **<zkNode_IP>:<zkNode_Port>** indica o URL do ZooKeeper. Use vírgulas (,) para separar vários URLs,

        exemplo: **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
      - **sparkthriftserver2x** indica o diretório do ZooKeeper, onde uma instância aleatória do JDBCServer está conectada ao cliente.

      Por exemplo, quando utiliza o cliente de Beeline para ligação, execute o seguinte comando:

      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;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.***<System domain name>***@***<System domain name>***;"**

      Se a autenticação Keytab estiver ativada, o CURL será o seguinte:

      ```
      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>
      ```

      No URL acima, *<principal_name>* indica o principal do usuário Kerberos, por exemplo, **test@***<System domain name>*; *<path_to_keytab>* indica o caminho do arquivo Keytab correspondente a *<principal_name>*, por exemplo, **/opt/auth/test/user.keytab**.

    - Modo comum:

      ```
      jdbc:hive2://
      <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:
      <zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=spa
      rkthriftserver2x;
      ```

      Por exemplo, execute o seguinte comando quando utiliza o cliente de Beeline para conexão no 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 não multi-locatário

  No modo não multi-locatário, um cliente se conecta a um nó do JDBCServer especificado. Comparado com o modo de instância multi-locatário, a cadeia de conexão neste modo não contém os parâmetros **serviceDiscoveryMode** e **zooKeeperNamespace** sobre ZooKeeper.

  Por exemplo, quando você usa o cliente de Beeline para conectar o JDBCServer no modo de instância não multi-locatário, execute o seguinte 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

  - No comando acima, **<server_IP>:<server_Port>** indica o URL do nó do JDBCServer especificado.
  - **CLIENT_HOME** indica o caminho do cliente.

  Exceto o método de conexão, outras operações da API do JDBCServer no modo multi-locatário e no modo não multi-locatário são as mesmas. Spark JDBCServer é outra implementação do HiveServer2 no Hive.Para detalhes sobre como usar o Spark JDBCServer, acesse o site oficial do Hive em **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

  Especificar um locatário

  Geralmente, o cliente enviado por um usuário se conecta ao JDBCServer padrão do locatário ao qual o usuário pertence. Se quiser conectar o cliente ao JDBCServer de um locatário especificado, adicione o parâmetro **--hiveconf mapreduce.job.queuename**.

  Se você usar o cliente de Beeline para conexão, execute o seguinte comando (**aaa** é o nome do locatário):

  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 Relação entre Spark, HDFS e Yarn

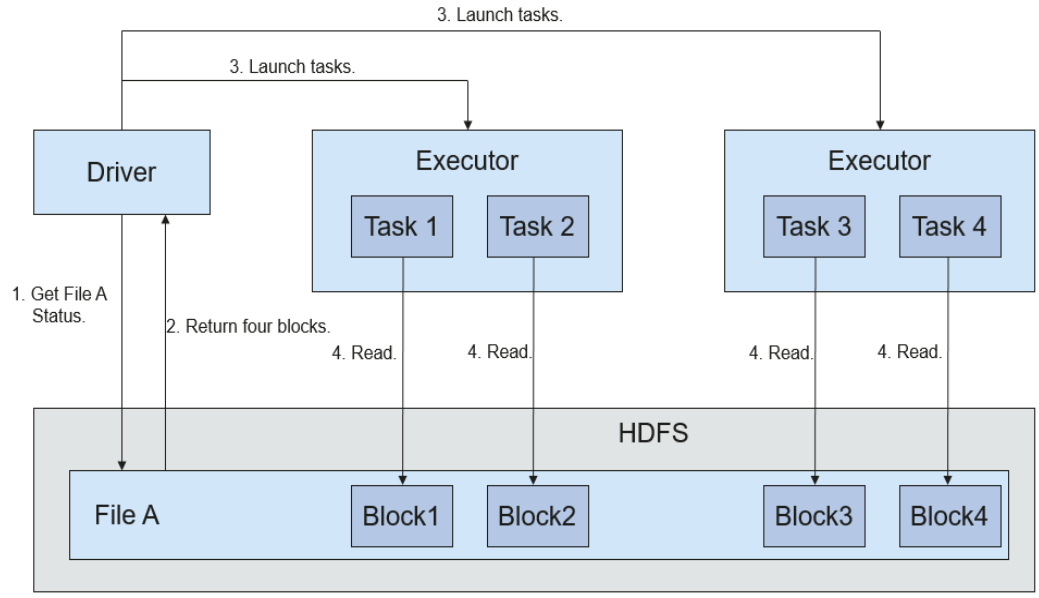
Relação entre Spark e HDFS

Os dados calculados pelo Spark vêm de várias fontes de dados, como arquivos locais e HDFS. A maioria dos dados computados pelo Spark vem do HDFS. O HDFS pode ler dados em grande escala para computação paralela. Depois de computados, os dados podem ser armazenados no HDFS.

O Spark envolve o Driver e o Executor. Driver agenda tarefas e o Executor executa tarefas.

Figura 6-98 mostra o processo de leitura de um arquivo.

Figura 6-98 Processo de leitura de arquivo

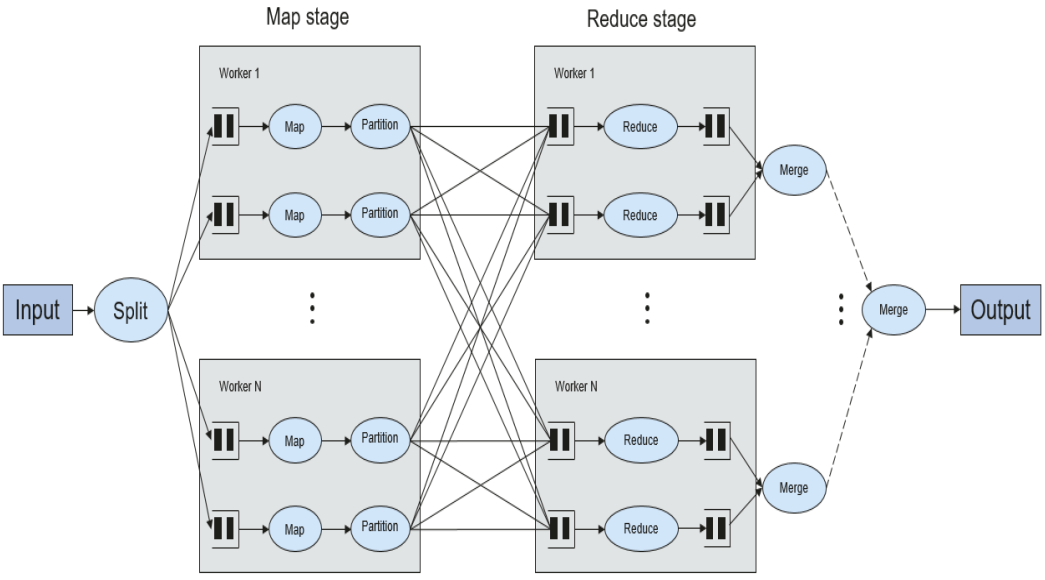


O processo de leitura do arquivo é o seguinte:

1. Driver se interconecta com o HDFS para obter as informações do Arquivo A.
2. O HDFS retorna as informações de bloco detalhadas sobre esse arquivo.
3. Driver define um grau paralelo com base na quantidade de dados do bloco e cria várias tarefas para ler os blocos desse arquivo.
4. Executor executa as tarefas e lê os blocos detalhados como parte do Conjunto de dados distribuído resiliente (RDD).

Figura 6-99 mostra o processo de gravação de dados em um arquivo.

Figura 6-99 Processo de gravação de arquivos



O processo de gravação de arquivos é o seguinte:

1. Driver cria um diretório onde o arquivo deve ser gravado.
2. Com base no status de distribuição do RDD, o número de tarefas relacionadas à gravação de dados é calculado e essas tarefas são enviadas ao Executor.
3. Executor executa essas tarefas e grava os dados do RDD no diretório criado em **1**.

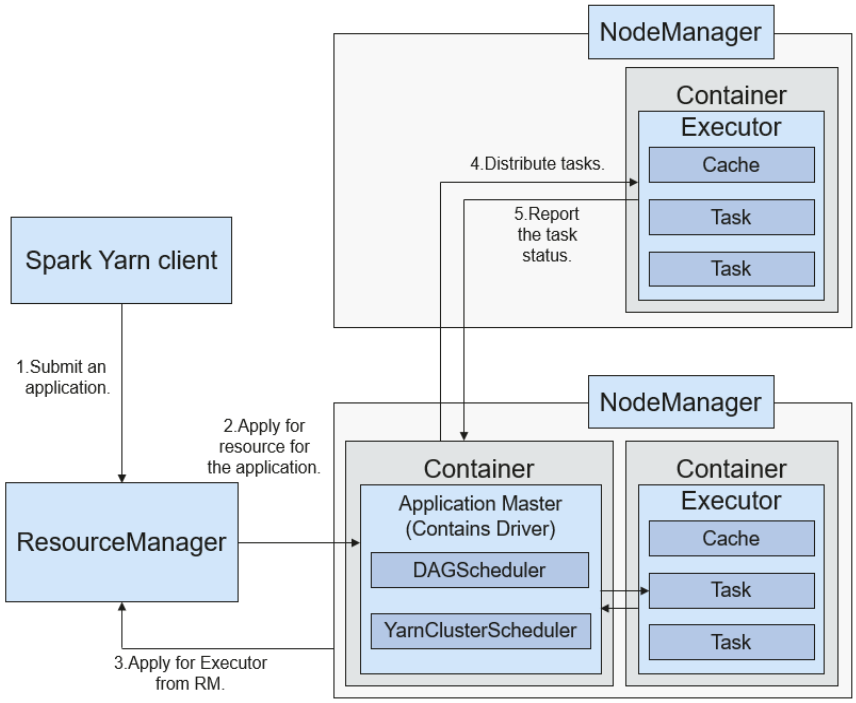
Relação entre Spark e Yarn

A computação e o agendamento do Spark podem ser implementados usando o modo Yarn. O Spark desfruta dos recursos de computação fornecidos pelos clusters de Yarn e executa tarefas de forma distribuída. Spark no Yarn tem dois modos: Yarn-cluster e Yarn-client.

- Modo Yarn-cluster

  Figura 6-100 mostra a estrutura em execução do Spark no Yarn-cluster.

  Figura 6-100 Estrutura de operação de Spark no Yarn-cluster



  Processo de implementação do Spark no Yarn-cluster:

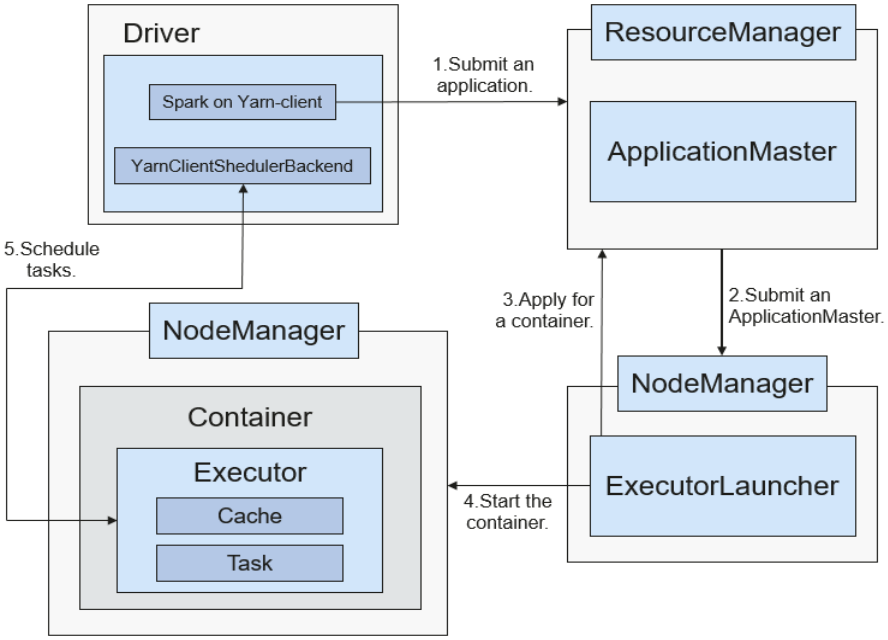
  a. O cliente gera as informações da aplicação e, em seguida, envia as informações para o ResourceManager.
  b. O ResourceManager aloca o primeiro contêiner (ApplicationMaster) para o SparkApplication e inicia o driver no contêiner.
  c. O ApplicationMaster se aplica a recursos do ResourceManager para executar o contêiner.

     O ResourceManager aloca o contêiner para o ApplicationMaster, que se comunica com o NodeManager, e inicia o executor no contêiner obtido. Depois que o executor é iniciado, ele se registra com o driver e solicita tarefas.
  d. O driver aloca tarefas para o executor.
  e. O executor executa tarefas e relata o status operacional para o driver.

- Modo Yarn-client

  Figura 6-101 mostra a estrutura em execução do Spark no Yarn-cluster.

  Figura 6-101 Estrutura de operação de Spark no Yarn-client



  Processo de implementação do Spark no Yarn-client:

  NOTA

  No modo Yarn-client, o Driver é implantado no cliente e iniciado no cliente. No modo cliente de fio, o cliente da versão anterior é incompatível. É aconselhável utilizar o modo Yarn-cluster.

  a. O cliente envia a solicitação da aplicação de Spark para ResourceManager e, em seguida, o ResourceManager retorna os resultados. Os resultados incluem informações como ID da solicitação e o máximo e mínimo de recursos disponíveis. O cliente empacota todas as informações necessárias para iniciar o ApplicationMaster e envia as informações para ResourceManager.
  b. Depois de receber a solicitação, o ResourceManager encontra um nó adequado para o ApplicationMaster e o inicia nesse nó. ApplicationMaster é uma função no Yarn e o nome do processo no Spark é ExecutorLauncher.
  c. Com base nos requisitos de recursos de cada tarefa, o ApplicationMaster pode solicitar uma série de Contêineres para executar tarefas do ResourceManager.
  d. Depois de receber a lista de contêineres recém-alocada (do ResourceManager), o ApplicationMaster envia informações aos NodeManagers relacionados para iniciar os contêineres.

     O ResourceManager aloca os contêineres para o ApplicationMaster, que se comunica com os NodeManagers relacionados e inicia os executores nos contêineres obtidos. Depois que os executores são iniciados, ele se registra com os drivers e aplica-se para as tarefas.

     NOTA

     Os contêineres em execução não são suspensos e os recursos não são liberados.

e. Os drivers alocam tarefas para os executores. O executor executa tarefas e relata o status de operação para o driver.

6.27.4 Recursos de código aberto aprimorados do Spark: consulta SQL otimizada de dados de origens diferentes

Cenário

As empresas geralmente armazenam dados massivos, como de vários bancos de dados e armazéns, para gerenciamento e coleta de informações. No entanto, fontes de dados diversificadas, estruturas de conjuntos de dados híbridos e armazenamento de dados dispersos reduzem a eficiência da consulta.

O Spark de código aberto só suporta pushdown de filtro simples durante a consulta de dados de várias fontes. O desempenho do mecanismo SQL está deteriorado devido a uma grande quantidade de transmissão de dados desnecessária. A função pushdown é aprimorada para que **aggregate**, **projection** complexa e **predicate** complexa possam ser enviados para fontes de dados, reduzindo a transmissão de dados desnecessária e melhorando o desempenho da consulta.

Somente a origem de dados JDBC suporta pushdown de operações de consulta, como **aggregate**, **projection**, **predicate**, **aggregate over inner join** e **aggregate over union all**. Todas as operações de pushdown podem ser ativadas com base em suas necessidades.

Tabela 6-23 Consulta aprimorada de consulta entre fontes

Módulo	Antes do aprimoramento	Após o aprimoramento
aggregate	O pushdown da **aggregate** não é suportado.	• Funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. Exemplo: selecione count(*) da tabela • Expressões internas de funções de agregação são suportadas. Exemplo: selecione sum(a+b) da tabela • O cálculo de funções de agregação é suportado. Exemplo: selecione avg(a) + max(b) da tabela • Pushdown de **having** é suportado. Exemplo: selecione sum(a) da tabela onde a>0 group by b tendo sum(a)>10 • O pushdown de algumas funções é suportado. Pushdown de linhas em matemática, tempo e funções de string, como **abs()**, **month()** e **length()** são suportados. Além das funções internas anteriores, você pode executar o comando **SET** para adicionar funções suportadas pelas fontes de dados. Exemplo: selecione sum(abs(a)) da tabela • Pushdown de **limit** e **order by** após **aggregate** é suportado. No entanto, o pushdown não é suportado no Oracle, porque o Oracle não suporta **limit**. Exemplo: selecione sum(a) da tabela onde a>0 group by b order by sum(a) limit 5
projection	Apenas pushdown de **projection** simples é suportado. Exemplo: selecione a, b da tabela	• Expressões complexas podem ser empurradas para baixo. Exemplo: selecione (a+b)*c da tabela • Algumas funções podem ser empurradas para baixo. Para obter detalhes, consulte a descrição abaixo da tabela. Exemplo: selecione length(a)+abs(b) da tabela • Pushdown do **limit** e **order by** após a **projection** é suportada. Exemplo: selecione a, b+c da tabela order by um limit 3

Módulo	Antes do aprimoramento	Após o aprimoramento
predicate	Somente a filtragem simples com o nome da coluna à esquerda do operador e os valores à direita é suportada. Exemplo: selecione * da tabela em que a>0 ou b em ("aaa", "bbb")	● O pushdown de expressões complexas é suportado. Exemplo: selecione * da tabela em que a +b>c*d ou a/c in (1, 2, 3) ● Algumas funções podem ser empurradas para baixo. Para obter detalhes, consulte a descrição abaixo da tabela. Exemplo: selecione * da tabela onde length(a)>5
aggregate over inner join	Os dados relacionados das duas tabelas devem ser carregados no Spark. A operação de junção deve ser executada antes da operação **aggregate**.	Há suporte para as seguintes funções: ● Funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. ● Todas as operações **aggregate** podem ser realizadas em uma mesma tabela. As operações **group by** podem ser executadas em uma ou duas tabelas e somente a junção interna é suportada. Os seguintes cenários não são suportados: ● **aggregate** não pode ser empurrado para baixo a partir das tabelas de junção esquerda e direita. ● **aggregate** contém operações, por exemplo, sum(a+b). ● Operações **aggregate**, por exemplo, sum(a)+min(b).
aggregate over union all	Os dados relacionados das duas tabelas devem ser carregados no Spark. **union** deve ser realizada antes de **aggregate**.	Cenários suportados: funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. Cenários não suportados: ● **aggregate** contém operações, por exemplo, sum(a+b). ● Operações **aggregate**, por exemplo, sum(a)+min(b).

Precauções

- Se a fonte de dados externa for Hive, a operação de consulta não poderá ser executada em tabelas externas criadas pelo Spark.
- Somente fontes de dados MySQL e MPPDB são suportadas.

6.28 Spark2x

6.28.1 Princípios básicos do Spark2x

NOTA

O componente Spark2x aplica-se ao MRS 3.x e versões posteriores.

Descrição

Spark é uma estrutura de computação distribuída baseada em memória. Em cenários iterativos, a capacidade computacional do Spark é de 10 a 100 vezes maior que a do MapReduce porque os dados são armazenados na memória ao serem processados. O Spark pode usar o HDFS como sistema de armazenamento subjacente, permitindo que os usuários mudem rapidamente do MapReduce para o Spark. O Spark fornece recursos de análise de dados completa, como o processamento de streaming em pequenos lotes, processamento em lote off-line, consulta SQL e mineração de dados. Os usuários podem usar perfeitamente essas funções em uma mesma aplicação. Para obter detalhes sobre os novos recursos de código aberto do Spark2x, consulte **Novos recursos de código aberto do Spark2x**.

Os recursos do Spark são os seguintes:

- Melhora a capacidade de processamento de dados por meio da computação de memória distribuída e do mecanismo de execução de grafo direcionado acíclico (DAG). O desempenho fornecido é de 10 a 100 vezes maior do que o do MapReduce.
- Suporta múltiplas linguagens de desenvolvimento (Scala/Java/Python) e dezenas de operadores altamente abstratos para facilitar a construção de aplicativos de processamento de dados distribuídos.
- Cria pilhas de processamento de dados usando **SQL**, **Streaming**, MLlib e GraphX para fornecer recursos de processamento de dados completo.
- Adapta-se ao ecossistema Hadoop, permitindo que aplicações Spark sejam executados em Standalone, Mesos ou Yarn, permitindo o acesso a várias fontes de dados, como HDFS, HBase e Hive e suportando a migração suave da aplicação MapReduce para Spark.

Arquitetura

Figura 6-102 descreve a arquitetura do Spark e **Tabela 6-24** lista os módulos do Spark.

Figura 6-102 Arquitetura do Spark

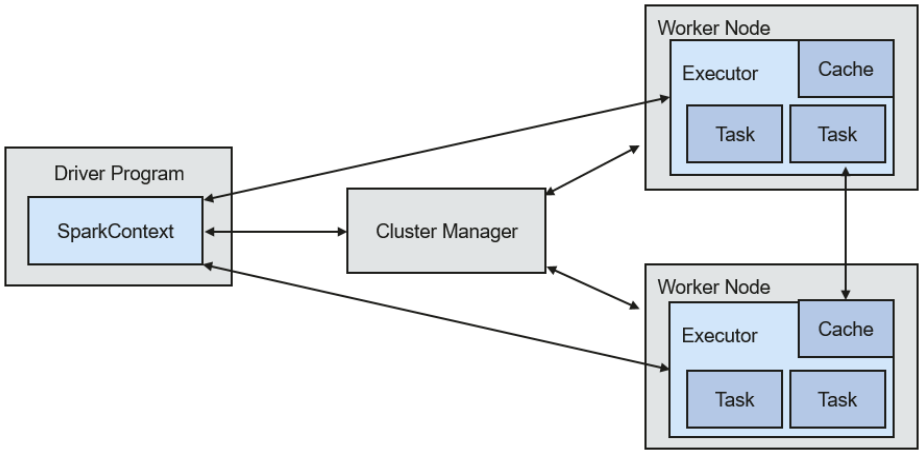


Tabela 6-24 Conceitos básicos

Módulo	Descrição
Gerenciador de cluster	O gerenciador de cluster gerencia os recursos no cluster. O Spark oferece suporte a vários gerenciadores de cluster, incluindo Mesos, Yarn e o gerenciador de cluster Standalone fornecido com o Spark. Por padrão, os clusters do Spark adotam o gerenciador de clusters do Yarn.
Aplicação	Aplicação Spark. Consiste em um Driver Program e vários executores.
Modo de implementação	Modo de implementação em cluster ou no cliente. No modo de cluster, o driver é executado em um nó dentro do cluster. No modo de cliente, o driver é executado no cliente (fora do cluster).
Programa do Driver	O processo principal da aplicação Spark. Ele executa a função **main()** de uma aplicação e cria SparkContext. Ele é usado para analisar aplicações, gerar estágios e agendar tarefas para executores. Normalmente, o SparkContext é o Programa de Drivers.
Executor	Um processo iniciado em um nó de trabalho. Ele é usado para executar tarefas e gerenciar e processar os dados usados em aplicações. Uma aplicação Spark geralmente contém vários executores. Cada executor recebe comandos do driver e executa uma ou várias tarefas.
Nó de trabalho	Um nó que inicia e gerencia executores e recursos em um cluster.
Job	Um job consiste em várias tarefas simultâneas. Um operador de ação (por exemplo, um operador de coleta) mapeia para um job.
Estágio	Cada job consiste em vários estágios. Cada estágio é um conjunto de tarefas, que é separado por Grafo Direcionado Acíclico (DAG).

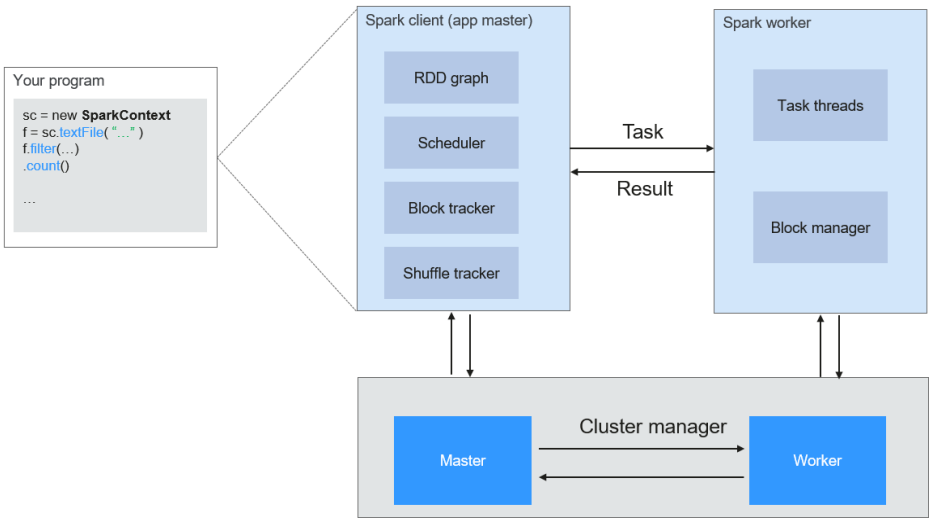
Módulo	Descrição
Tarefa	Uma tarefa carrega a unidade de computação das lógicas de serviço. É a unidade de trabalho mínima que pode ser executada na plataforma Spark. Uma aplicação pode ser dividida em várias tarefas com base no plano de execução e na quantidade de computação.

Princípio do Spark

Figura 6-103 descreve a arquitetura de execução da aplicação do Spark.

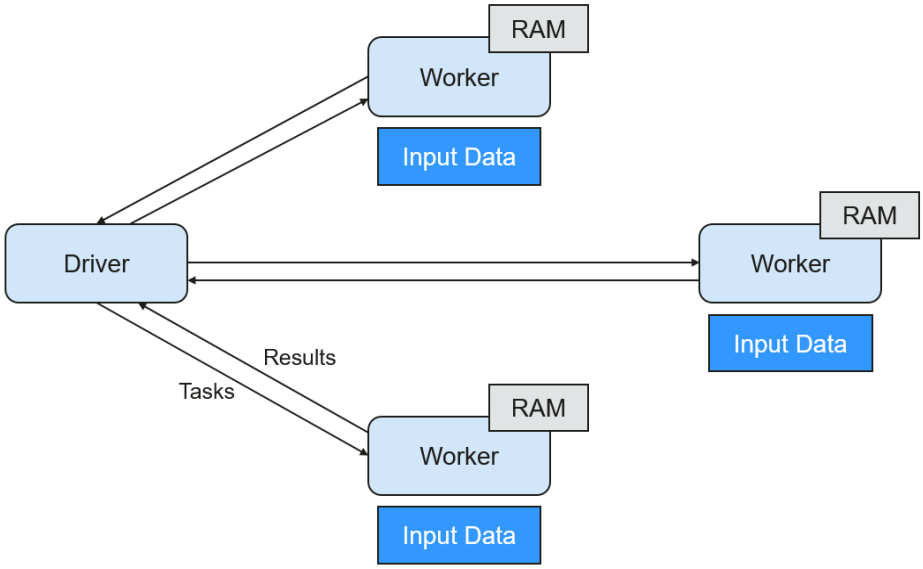
1. Uma aplicação está sendo executada no cluster como uma coleção de processos. Driver coordena a execução da aplicação.
2. Para executar uma aplicação, o Driver se conecta ao gerenciador de cluster (como Standalone, Mesos e Yarn) para solicitar os recursos do executor e iniciar o ExecutorBackend. O gerenciador de cluster agenda recursos entre diferentes aplicações. Driver agenda DAGs, divide estágios e gera tarefas para a aplicação ao mesmo tempo.
3. Em seguida, o Spark envia os códigos da aplicação (os códigos transferidos para **SparkContext**, que são definidos por JAR ou Python) a um executor.
4. Depois que todas as tarefas forem concluídas, a execução da aplicação do usuário é interrompida.

Figura 6-103 Arquitetura de execução da aplicação Spark



O Spark usa os modos Master e Worker, como mostrado na **Figura 6-104**. Um usuário envia uma aplicação no cliente do Spark e, em seguida, o agendador divide um job em várias tarefas e envia as tarefas a cada Worker para execução. Cada Worker reporta os resultados do cálculo para o Driver (Master) e, em seguida, o Driver agrega e retorna os resultados para o cliente.

Figura 6-104 Modo Master-Worker do Spark



Observe o seguinte sobre a arquitetura:

- As aplicações são isoladas umas das outras.

  Cada aplicação tem um processo de executor independente, e cada executor inicia vários threads para executar tarefas em paralelo. Cada driver agenda suas próprias tarefas e diferentes tarefas de aplicação são executadas em diferentes JVMs, ou seja, diferentes executores.
- Diferentes aplicações do Spark não compartilham dados, a menos que os dados sejam armazenados no sistema de armazenamento externo, como o HDFS.
- É aconselhável implementar o programa do Driver em um local próximo ao nó de trabalho porque o programa do Driver agenda tarefas no cluster. Por exemplo, implante o programa do Driver na rede onde o nó de trabalho está localizado.

O Spark no YARN pode ser implementado em dois modos:

- No modo Yarn-cluster, o driver do Spark é executado dentro de um processo ApplicationMaster que é gerenciado pelo Yarn no cluster. Depois que o ApplicationMaster é iniciado, o cliente pode sair sem interromper a execução do serviço.
- No modo Yarn-client, o Driver é executado no processo de cliente e o processo ApplicationMaster é usado apenas para solicitar recursos do Yarn.

Princípio do Spark Streaming

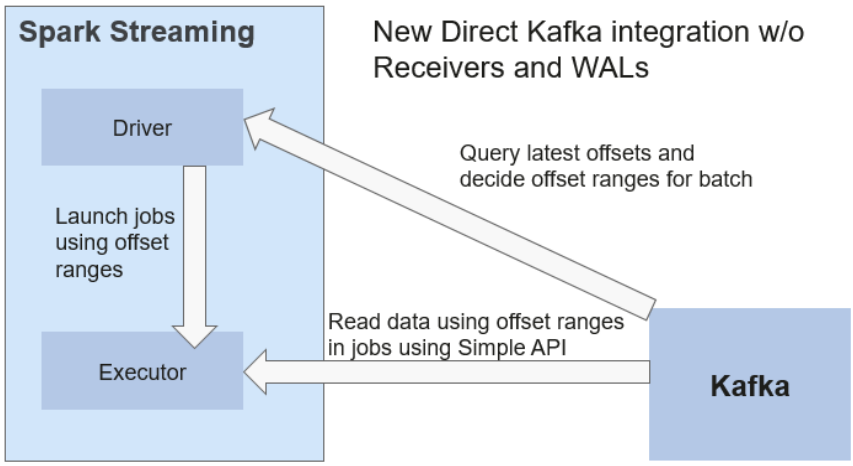
O Spark Streaming é uma estrutura de computação em tempo real construída sobre o Spark, que expande a capacidade de processar dados de streaming em massa. O Spark suporta duas abordagens de processamento de dados: Streaming direto e Receptor.

Processo de computação de Streaming direto

Na abordagem de Streaming direto, a API direta é usada para processar dados. Tomemos como exemplo a API direta do Kafka. A API direta fornece o local de deslocamento do qual cada intervalo de lote será lido, o que é muito mais simples do que iniciar um receptor para receber continuamente dados do Kafka e dados gravados em logs de gravação antecipada (WALs). Em seguida, cada job em lote está em execução e os dados de deslocamento

correspondentes estão prontos no Kafka. Essas informações de deslocamento podem ser armazenadas com segurança no arquivo de ponto de verificação e lidas por aplicações que falharam ao iniciar.

Figura 6-105 Transmissão de dados através da API direta do Kafka



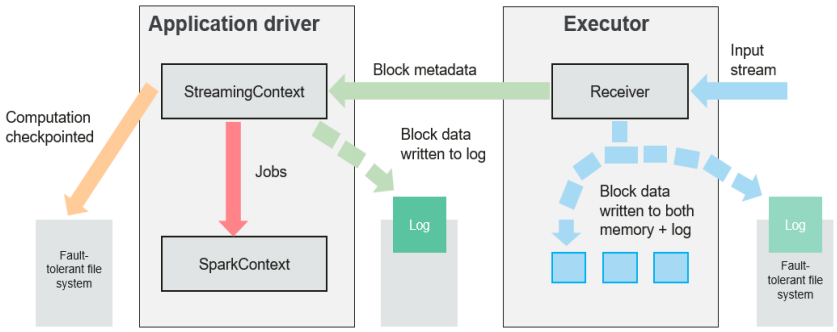
Após a falha, o Spark Streaming pode ler dados do Kafka novamente e processar o segmento de dados. O resultado do processamento é o mesmo, não importa se o Spark Streaming falhar ou não, porque a semântica é processada apenas uma vez.

A API direta não precisa usar o WAL e os receptores e garante que cada registro do Kafka seja recebido apenas uma vez, o que é mais eficiente. Desta forma, o Spark Streaming e o Kafka podem ser bem integrados, fazendo com que os canais de streaming sejam apresentados com alta tolerância a falhas, alta eficiência e facilidade de uso. Portanto, é aconselhável usar o Streaming direto para processar dados.

Processo de computação do Receptor

Quando uma aplicação Spark Streaming é iniciada, o StreamingContext relacionado (a base de todas as funções de streaming) usa o SparkContext para iniciar o receptor e se tornar uma tarefa de longa duração. Esses receptores recebem e salvam dados de streaming na memória do Spark para processamento. **Figura 6-106** mostra o ciclo de vida da transferência de dados.

Figura 6-106 Ciclo de vida da transferência de dados



1. Receber dados (seta azul).

O receptor divide um fluxo de dados em uma série de blocos e os armazena na memória do executor. Além disso, depois que a WAL é habilitada, ela grava dados na WAL do sistema de arquivos tolerante a falhas.

2. Notificar o driver (seta verde).

   Os metadados no bloco recebido são enviados para StreamingContext no driver. Os metadados incluem:

   - ID de referência de bloco usado para localizar a posição dos dados na memória do Executor.
   - Bloquear informações de deslocamento de dados em logs (se a função WAL estiver ativada).

3. Dados do processo (seta vermelha).

   Para cada lote de dados, o StreamingContext usa informações de bloco para gerar conjuntos de dados distribuídos resilientes (RDDs) e jobs. O StreamingContext executa jobs executando tarefas para processar blocos na memória do executor.

4. Periodicamente definir pontos de verificação (setas laranja).
5. Para tolerância a falhas, o StreamingContext define periodicamente pontos de verificação e os salva em sistemas de arquivos externos.

Tolerância a falhas

O Spark e seu RDD permitem o processamento contínuo de falhas de qualquer nó de trabalho no cluster. O Spark Streaming é construído em cima do Spark. Portanto, o nó de trabalho do Spark Streaming também tem a mesma capacidade de tolerância a falhas. No entanto, o Spark Streaming precisa ser executado corretamente em caso de execução de longo tempo. Portanto, o Spark deve ser capaz de se recuperar de falhas através do processo de driver (processo principal que coordena todos os Workers). Isso representa desafios para a tolerância a falhas do driver do Spark, pois o driver do Spark pode ser qualquer aplicação de usuário implementado em qualquer modo de computação. No entanto, o Spark Streaming possui uma arquitetura de computação interna. Ou seja, ele executa periodicamente a mesma computação do Spark em cada dado de lote. Tal arquitetura permite que ele armazene periodicamente pontos de verificação para espaço de armazenamento confiável e os recupere após a reinicialização do Driver.

Para dados de origem, como arquivos, o mecanismo de recuperação de Driver pode garantir zero perda de dados, pois todos os dados são armazenados em um sistema de arquivos tolerante a falhas, como o HDFS. No entanto, para outras fontes de dados, como Kafka e Flume, alguns dados recebidos são armazenados em cache apenas na memória e podem ser perdidos antes de serem processados. Isso é causado pelo modo de operação de distribuição das aplicações Spark. Quando o processo de driver falha, todos os executores em execução no Gerenciador de Cluster, juntamente com todos os dados na memória, são encerrados. Para evitar essa perda de dados, a função WAL é adicionada ao Spark Streaming.

O WAL é frequentemente usado em bancos de dados e sistemas de arquivos para garantir a persistência de qualquer operação de dados. Ou seja, primeiro registre uma operação em um log persistente e execute essa operação nos dados. Se a operação falhar, o sistema é recuperado lendo o log e reaplicando a operação predefinida. A seguir, descrevemos como usar o WAL para garantir a persistência dos dados recebidos:

Receptor é usado para receber dados de fontes de dados como Kafka. Como uma tarefa de longa duração no Executor, o Receptor recebe dados e também confirma os dados recebidos se suportados por fontes de dados. Os dados recebidos são armazenados na memória do Executor e o Driver entrega uma tarefa ao Executor para processamento.

Depois que o WAL é habilitado, todos os dados recebidos são armazenados em arquivos de log no sistema de arquivos tolerante a falhas. Portanto, os dados recebidos não são perdidos mesmo se o Spark Streaming falhar. Além disso, o receptor verifica a exatidão dos dados recebidos somente depois que os dados são pré-gravados em logs. Os dados que estão em cache, mas não armazenados, podem ser enviados novamente por fontes de dados após a reinicialização do driver. Esses dois mecanismos garantem perda zero de dados. Ou seja, todos os dados são recuperados de logs ou reenviados por fontes de dados.

Para ativar a função WAL, execute as seguintes operações:

- Defina **streamingContext.checkpoint** (caminho para diretório) para configurar o diretório de ponto de verificação, que é um caminho de arquivo HDFS usado para armazenar pontos de verificação de streaming e WALs.
- Defina **spark.streaming.receiver.writeAheadLog.enable** de SparkConf para **true** (o valor padrão é **false**).

Depois que o WAL é ativado, todos os receptores têm a vantagem de se recuperar de dados recebidos confiáveis. É aconselhável desabilitar o mecanismo de várias réplicas porque o sistema de arquivos tolerante a falhas do WAL também pode replicar os dados.

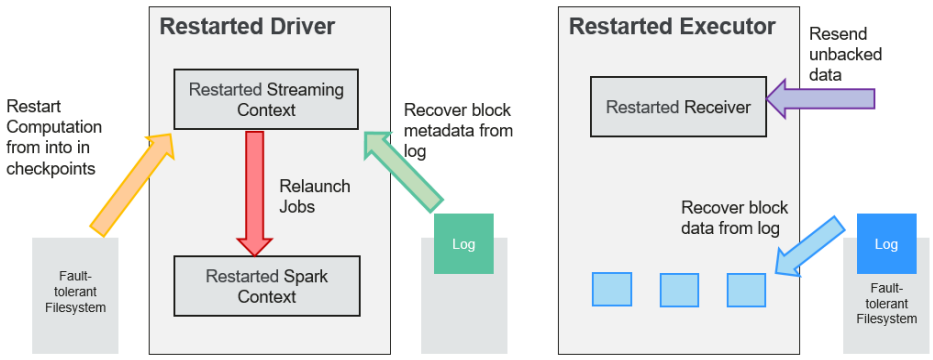
NOTA

A taxa de transferência de recebimento de dados é reduzida depois que o WAL é habilitado. Todos os dados são gravados no sistema de arquivos tolerante a falhas. Como resultado, a taxa de transferência de gravação do sistema de arquivos e a largura de banda da rede para replicação de dados podem se tornar o gargalo potencial. Para resolver esse problema, é aconselhável criar mais receptores para aumentar o grau de paralelismo de recebimento de dados ou usar um hardware melhor para melhorar a taxa de transferência do sistema de arquivos tolerante a falhas.

Processo de recuperação

Quando um driver com falha for reiniciado, reinicie-o da seguinte maneira:

Figura 6-107 Processo de recuperação de computação



1. Recupere a computação. (Seta laranja)

   Use as informações do ponto de verificação para reiniciar o Driver, reconstruir o SparkContext e reiniciar o Receptor.

2. Recupere bloco de metadados. (Seta verde)

   Essa operação garante que todos os blocos de metadados necessários sejam recuperados para continuar a recuperação computacional subsequente.

3. Relance jobs inacabados. (Seta vermelha)

Os metadados recuperados são usados para gerar RDDs e trabalhos correspondentes para processamento em lote interrompido devido a falhas.

4. Leia dados de bloco salvos em logs. (Seta azul)

   Os dados de bloco são lidos diretamente dos WALs durante a execução dos trabalhos anteriores e, portanto, todos os dados essenciais armazenados de forma confiável nos logs são recuperados.

5. Reenvie dados não confirmados. (Seta roxa)

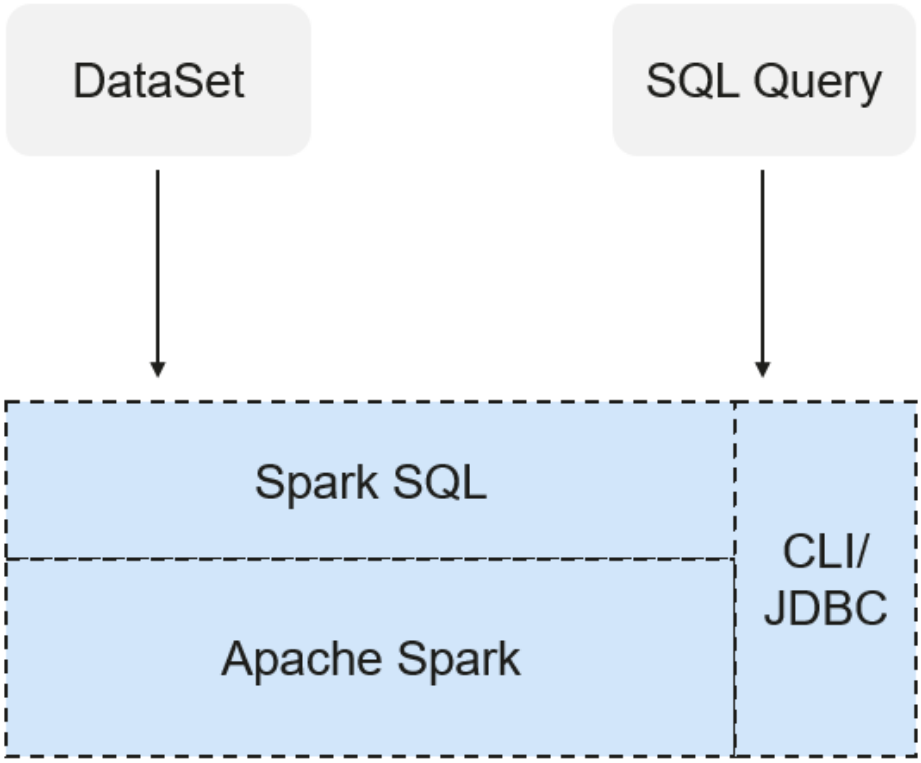
   Os dados que são armazenados em cache, mas não armazenados em logs após falhas, são reenviados por fontes de dados, porque o receptor não confirma os dados.

Portanto, usando WALs e um Receptor confiável, o Spark Streaming pode evitar a perda de dados de entrada causada por falhas de Driver.

Princípio do SparkSQL e do DataSet

SparkSQL

Figura 6-108 SparkSQL e DataSet



Spark SQL é um módulo para processamento de dados estruturados. Na aplicação Spark, instruções SQL ou APIs de DataSet podem ser usadas para consultar dados estruturados.

Spark SQL e DataSet também fornecem um método universal para acessar várias fontes de dados, como Hive, CSV, Parquet, ORC, JSON e JDBC. Essas fontes de dados também permitem a interação de dados. O Spark SQL reutiliza a lógica de processamento de front-end do Hive e o módulo de processamento de metadados. Com o Spark SQL, você pode consultar diretamente os dados existentes do Hive.

Além disso, o Spark SQL também fornece APIs, CLI e APIs de JDBC, permitindo diversos acessos ao cliente.

DDL/DML do Spark SQL nativo

No Spark 1.5, muitos comandos de Linguagem de Definição de Dados (DDL)/Linguagem de Manipulação de Dados (DML) são empurrados para baixo e executados no Hive, causando acoplamento com o Hive e inflexibilidade, como relatórios e resultados de erros inesperados.

O Spark2x realiza a localização de comandos e substitui o Hive por DDL/DML de Spark SQL Nativo para executar comandos DDL/DML. Além disso, o desacoplamento do Hive é realizado e os comandos podem ser personalizados.

DataSet

Um DataSet é uma coleção fortemente tipada de objetos específicos de domínio que podem ser transformados em paralelo usando operações funcionais ou relacionais. Cada Dataset também tem uma exibição não tipada chamada DataFrame que é um Dataset de linha.

O DataFrame é um conjunto de dados estruturado e distribuído que consiste em várias colunas. O DataFrame é igual a uma tabela no banco de dados de relacionamento ou o DataFrame no R/Python. O DataFrame é o conceito mais básico do Spark SQL, que pode ser criado usando vários métodos, como o conjunto de dados estruturado, a tabela de Hive, o banco de dados externo ou o RDD.

As operações disponíveis no DataSets são divididas em transformações e ações.

- Uma operação de transformação pode gerar um novo DataSet,

  por exemplo, **map**, **filter**, **select** e **aggregate (groupBy)**.
- Uma operação de ação pode acionar a computação e retornar resultados,

  por exemplo, **count**, **show** ou gravar dados no sistema de arquivos.

Você pode usar um dos seguintes métodos para criar um DataSet:

- A maneira mais comum é apontar o Spark para alguns arquivos em sistemas de armazenamento, usando a função **read** disponível em um SparkSession.
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people =
  spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- Você também pode criar um DataSet usando a operação de transformação disponível em um existente. Por exemplo, aplique a operação de mapa em um DataSet existente para criar um DataSet:
  ```
  val names = people.map(_.name) // In Scala: names is Dataset.
  Dataset<String> names = people.map((Person p) -> p.name,
  Encoders.STRING)); // Java
  ```

CLI e JDBCServer

Além das APIs de programação, o Spark SQL também fornece as APIs de CLI/JDBC.

- Os scripts **spark-shell** e **spark-sql** podem fornecer a CLI para depuração.
- JDBCServer fornece APIs de JDBC. Sistemas externos podem enviar diretamente solicitações de JDBC para calcular e analisar dados estruturados.

Princípio de SparkSession

SparkSession é uma API unificada no Spark2x e pode ser considerada como uma entrada unificada para leitura de dados. SparkSession fornece um único ponto de entrada para

executar muitas operações anteriormente espalhadas por várias classes e também fornece métodos de acesso a essas classes antigas para maximizar a compatibilidade.

Uma SparkSession pode ser criado usando um padrão de construtor. O construtor reutilizará automaticamente a SparkSession existente se houver uma SparkSession ou criará uma SparkSession se ela não existir. Durante as transações de I/O, as definições do item de configuração no construtor são automaticamente sincronizadas com o Spark e o 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()
```

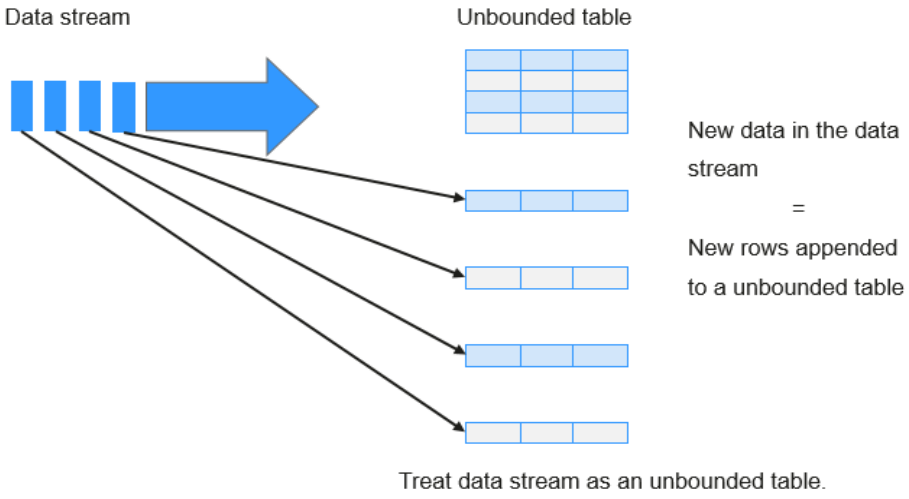
- SparkSession pode ser usada para executar consultas SQL em dados e retornar resultados como DataFrame.
  ```
  sparkSession.sql("select * from person").show
  ```
- A SparkSession pode ser usada para definir itens de configuração durante a execução. Esses itens de configuração podem ser substituídos por variáveis em instruções SQL.
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- A SparkSession também inclui um método "catalog" que contém métodos para trabalhar com o Metastore (catálogo de dados). Depois que esse método é usado, um conjunto de dados é retornado, que pode ser executado usando a mesma API de Dataset.
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```
- O SparkContext subjacente pode ser acessado pela API de SparkContext da SparkSession.
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Princípio do Streaming Estruturado

O Streaming Estruturado é um mecanismo de processamento de fluxo construído no motor Spark SQL. Você pode usar a API de Dataset/DataFrame em Scala, Java, Python ou R para expressar agregações de streaming, janelas de tempo de evento e junções fluxo-fluxo. Se os dados de streaming forem produzidos de forma incremental e contínua, o Spark SQL continuará processando os dados e sincronizando o resultado com o conjunto de resultados. Além disso, o sistema garante de ponta a ponta garantias de tolerância a falhas exatas através de pontos de verificação e WALs.

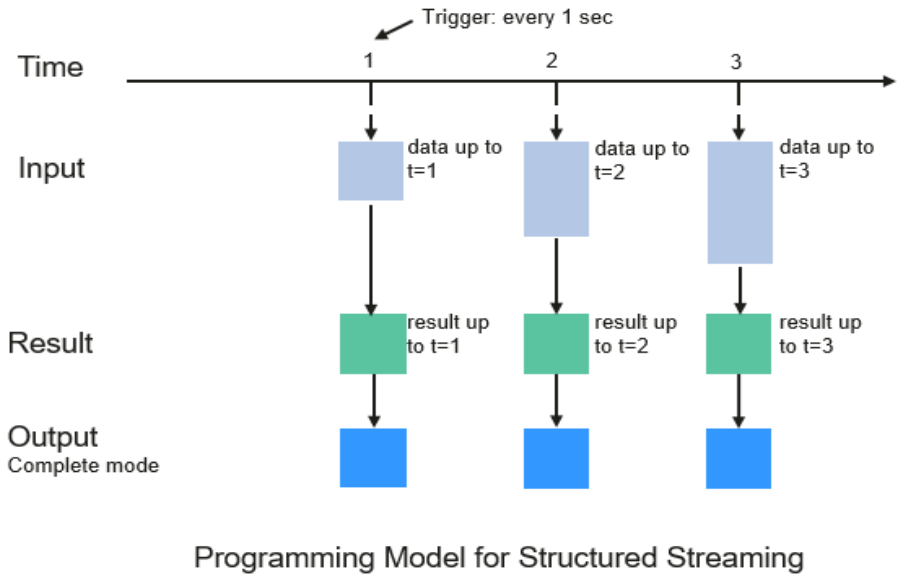
O núcleo do Streaming Estruturado é tomar dados de streaming como uma tabela de banco de dados incremental. Semelhante ao modelo de processamento de blocos de dados, o modelo de processamento de dados de streaming aplica operações de consulta em uma tabela de banco de dados estática à computação de streaming, e o Spark usa instruções SQL padrão para consulta, para obter dados da tabela incremental e ilimitada.

Figura 6-109 Tabela ilimitada de Streaming Estruturado



Cada operação de consulta gera uma tabela de resultados. Em cada intervalo de gatilho, os dados atualizados serão sincronizados com a tabela de resultados. Sempre que a tabela de resultados for atualizada, o resultado atualizado será gravado em um sistema de armazenamento externo.

Figura 6-110 Modelo de processamento de dados de Streaming Estruturado



Os modos de armazenamento de Streaming Estruturado na fase de saída são os seguintes:

- Modo completo: os conjuntos de resultados atualizados são gravados no sistema de armazenamento externo. A operação de gravação é realizada por um conector do sistema de armazenamento externo.

- Modo de adição: se um intervalo for acionado, apenas os dados adicionados na tabela de resultados serão gravados em um sistema externo. Isso é aplicável somente nas consultas em que não se espera que as linhas existentes na tabela de resultados sejam alteradas.
- Modo de atualização: se um intervalo for acionado, somente os dados atualizados na tabela de resultados serão gravados em um sistema externo, que é a diferença entre o Modo completo e o Modo de atualização.

Conceitos

- **RDD**

  Conjunto de dados distribuídos resilientes (RDD) é um conceito central do Spark. Indica um conjunto de dados distribuído somente leitura e particionado. Parciais ou todos os dados deste conjunto de dados podem ser armazenados em cache na memória e reutilizados entre cálculos.

  Criação de RDD

  - Um RDD pode ser criado a partir da entrada do HDFS ou de outros sistemas de armazenamento compatíveis com o Hadoop.
  - Um novo RDD pode ser convertido a partir de um RDD pai.
  - Um RDD pode ser convertido a partir de uma coleção de conjuntos de dados através da codificação.

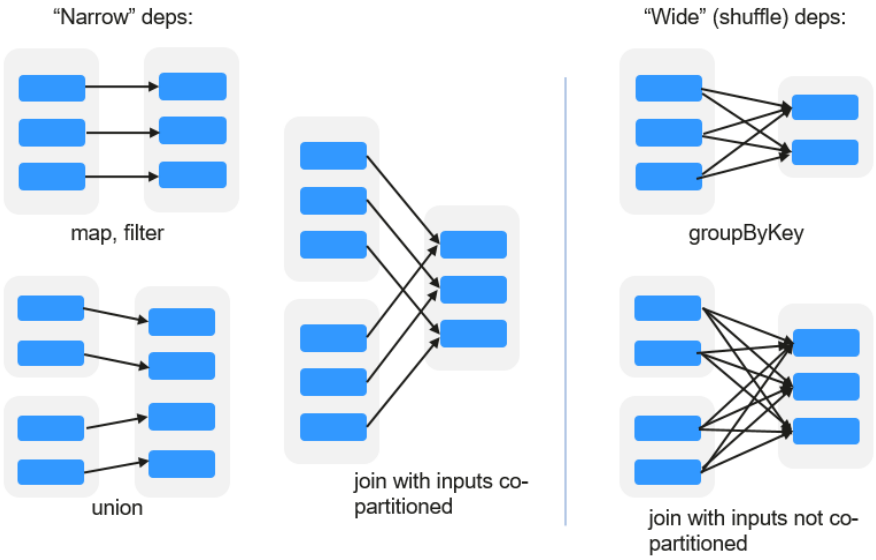
  Armazenamento de RDD

  - Você pode selecionar diferentes níveis de armazenamento para armazenar um RDD para reutilização. (Há 11 níveis de armazenamento para armazenar um RDD.)
  - Por padrão, o RDD é armazenado na memória. Quando a memória é insuficiente, o RDD transborda para o disco.

- **Dependência de RDD**

  A dependência de RDD inclui a dependência estreita e a dependência ampla.

  Figura 6-111 Dependência de RDD

  - **Dependência estreita**: cada partição do RDD pai é usada por no máximo uma partição do RDD filho.

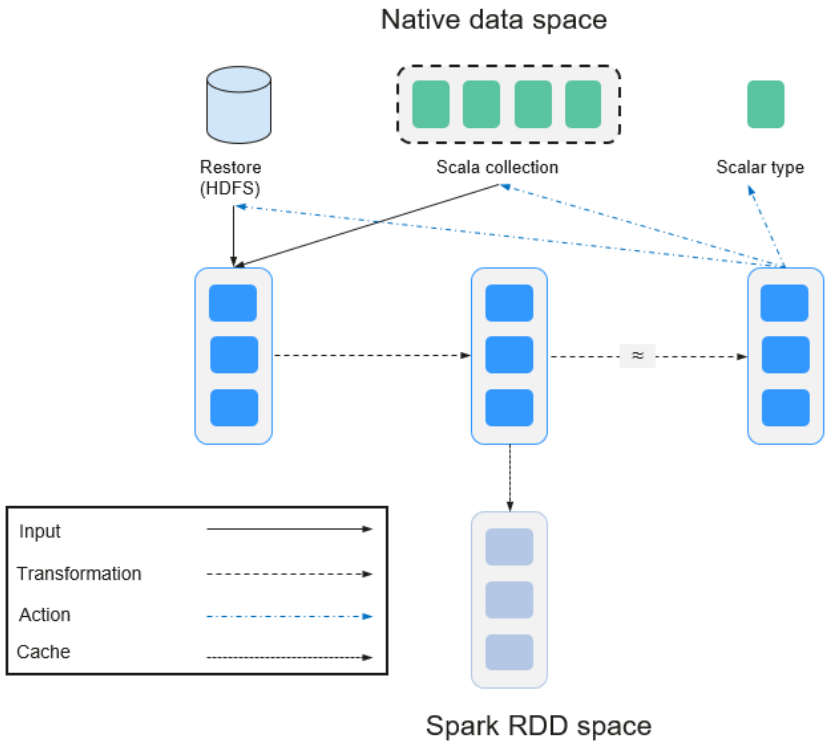
- **Dependência ampla**: as partições do RDD filho dependem de todas as partições do RDD pai.

A dependência estreita facilita a otimização. Logicamente, cada operador do RDD é um fork/join (a join não é o operador de junção mencionado acima, mas a barreira usada para sincronizar várias tarefas simultâneas); fork o RDD para cada partição e, em seguida, execute a computação. Após a computação, junte os resultados e, em seguida, execute a operação fork/join no próximo operador do RDD. É antieconômico traduzir diretamente o RDD em implementação física. A primeira é que cada RDD (mesmo resultado intermediário) precisa ser fisicalizado em memória ou armazenamento, o que é demorado e ocupa muito espaço. A segunda é que, como uma barreira global, a operação join é muito cara e todo o processo de join será retardado pelo nó mais lento. Se as partições do RDD filho dependerem estritamente das do RDD pai, os dois processos de fork/join podem ser combinados para implementar a otimização de fusão clássica. Se a relação na sequência contínua do operador for de dependência estreita, vários processos de fork/join podem ser combinados para reduzir um grande número de barreiras globais e eliminar a fisicalização de muitos resultados intermediários do RDD, o que melhora muito o desempenho. Isso é chamado de otimização de pipeline no Spark.

- **Transformação e ação (operações de RDD)**

  As operações no RDD incluem transformação (o valor de retorno é um RDD) e ação (o valor de retorno não é um RDD). **Figura 6-112** mostra o processo de operação do RDD. A transformação é lenta, o que indica que a transformação de um RDD para outro RDD não é executada imediatamente. O Spark registra apenas a transformação, mas não a executa imediatamente. A computação real é iniciada somente quando a ação é iniciada. A ação retorna resultados ou grava os dados do RDD no sistema de armazenamento. A ação é a força motriz para o Spark iniciar a computação.

Figura 6-112 Operação de RDD



Os dados e o modelo de operação do RDD são bem diferentes dos do Scala.


```
val file = sc.textFile("hdfs://...")
val errors = file.filter(_.contains("ERROR"))
errors.cache()
errors.count()
```

a. O operador de textFile lê arquivos de log do HDFS e retorna arquivos (como um RDD).

b. O operador de filtro filtra as linhas com **ERROR** e as atribui a erros (um novo RDD). O operador de filtro é uma transformação.

c. O operador de cache armazena em cache os erros para uso futuro.

d. O operador de contagem retorna o número de linhas de erros. O operador de contagem é uma ação.

A transformação inclui os seguintes tipos:

- Os elementos do RDD são considerados elementos simples.

  A entrada e a saída têm a relação um-para-um, e a estrutura de partição do RDD resultante permanece inalterada, por exemplo, map.

  A entrada e a saída têm a relação um-para-muitos, e a estrutura de partição do RDD resultante permanece inalterada, por exemplo, flatMap (um elemento torna-se uma sequência contendo vários elementos após map e, em seguida, achata para vários elementos).

  A entrada e a saída têm a relação um-para-um, mas a estrutura da partição do RDD resultante muda, por exemplo, union (dois RDDs integram-se a um RDD, e o número de partições torna-se a soma do número de partições de dois RDDs) e coalesce (as partições são reduzidas).

  Operadores de alguns elementos são selecionados a partir da entrada, como filter, distinct (elementos duplicados são excluídos), subtract (elementos que só existem neste RDD são mantidos) e sample (amostras são tomadas).

- Os elementos do RDD são considerados pares chave-valor.

  Executar o cálculo um-para-um no RDD único, como mapValues (o modo de partição do RDD de origem é mantido, o que é diferente de map).

  Classificar o RDD único, como sort e particionarPor (particionamento com consistência, o que é importante para a otimização local).

  Reestruturar e reduzir o RDD único com base na chave, como groupByKey e reduceByKey.

  Juntar-se e reestruturar dois RDDs com base na chave, como join e cogroup.

  NOTA

  As três operações posteriores que envolvem ordenação são chamadas de operações de shuffle.

A ação inclui os seguintes tipos:

- Gerar itens de configuração de escalar, como **count** (o número de elementos no RDD retornado), **reduce**, **fold/aggregate** (o número de itens de configuração de escalar que são retornados) e **take** (o número de elementos antes do retorno).
- Gerar a coleção Scala, como **collect** (importar todos os elementos do RDD para a coleção Scala) e **lookup** (procurar todos os valores correspondentes à chave).
- Gravar dados no armazenamento, como **saveAsTextFile** (que corresponde ao **textFile** anterior).
- Pontos de verificação, como o operador do **checkpoint**. Quando o Lineage é bastante longo (o que ocorre frequentemente na computação gráfica), leva um longo

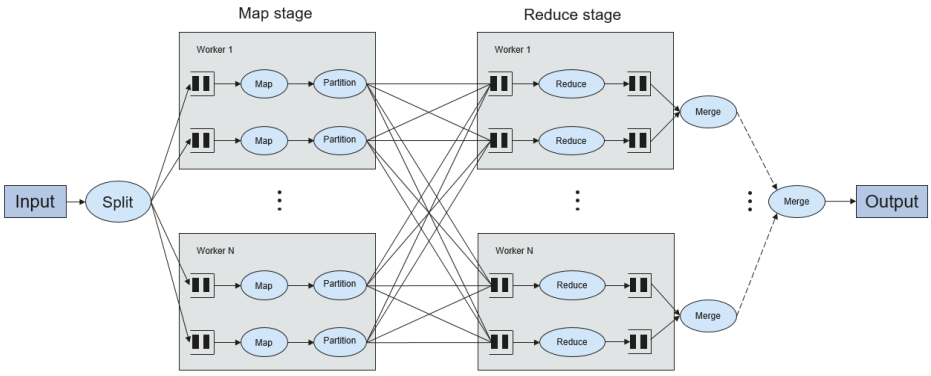
período de tempo para executar toda a sequência novamente quando ocorre uma falha. Nesse caso, o ponto de verificação é usado como o ponto de verificação para gravar os dados atuais no armazenamento estável.

- **Shuffle**

  Shuffle é uma fase específica na estrutura MapReduce, que está localizada entre a fase Map e a fase Reduce. Se os resultados de saída de Map forem usados pela Reduce, os resultados de saída deverão ser hash com base em uma chave e distribuídos para cada Redutor. Esse processo é chamado de Shuffle. Shuffle envolve a leitura e a escrita do disco e a transmissão da rede, de modo que o desempenho de Shuffle afete diretamente a eficiência da operação de todo o programa.

  A figura abaixo mostra todo o processo do algoritmo do MapReduce.

  Figura 6-113 Processo de algoritmo

  Shuffle é uma ponte para conectar dados. A seguir, descreve-se a implementação do shuffle no Spark.

  Shuffle divide um job do Spark em vários estágios. Os estágios anteriores contêm uma ou mais ShuffleMapTasks, e o último estágio contém uma ou mais ResultTasks.

- **Estrutura da aplicação Spark**

  A estrutura da aplicação Spark inclui o SparkContext inicializado e o programa principal.

  - SparkContext inicializado: constrói o ambiente operacional da aplicação Spark.

    Constrói o objeto do SparkContext. O seguinte é um exemplo:

    ```
    new SparkContext(master, appName, [SparkHome], [jars])
    ```

    Descrição do parâmetro:

    master: indica a cadeia de ligação. Os modos de link incluem local, Yarn-cluster e Yarn-client.

    appName: indica o nome da aplicação.

    SparkHome: indica o diretório em que o Spark está instalado no cluster.

    jars: indica o pacote de código e dependência de uma aplicação.

  - Programa principal: processa dados.

  Para obter detalhes sobre como enviar uma inscrição, visite **https://spark.apache.org/docs/3.1.1/submitting-applications.html**.

- **Comandos Shell do Spark**

  Os comandos básicos shell do Spark suportam o envio de aplicações Spark. Os comandos shell do Spark são os seguintes:


```
                        ./bin/spark-submit \
                        --class <main-class> \
                        --master <master-url> \
... # other options
<application-jar> \
[application-arguments]
```

Descrição do parâmetro:

--class: indica o nome da classe de uma aplicação Spark.

--master: indica o mestre ao qual o aplicativo Spark se vincula, como Yarn-client e Yarn-cluster.

application-jar: indica o caminho do arquivo JAR da aplicação Spark.

application-arguments: indica o parâmetro necessário para enviar a aplicação Spark. Este parâmetro pode ser deixado em branco.

- **Servidor JobHistory do Spark**

  A IU da Web do Spark é usada para monitorar os detalhes em cada fase da estrutura do Spark de um job do Spark em execução ou histórico e fornecer a exibição do log, o que ajuda os usuários a desenvolver, configurar e otimizar o job em unidades mais refinadas.

6.28.2 Solução HA do Spark2x

6.28.2.1 Instância multi-ativa do Spark2x

Conhecimento de fundo

Com base nos JDBCServers existentes na comunidade, o HA de instâncias multi-ativas é usado para alcançar a alta disponibilidade. Neste modo, vários JDBCServers coexistem no cluster e o cliente pode conectar aleatoriamente qualquer JDBCServer para executar operações de serviço. Quando um ou vários JDBCServers param de funcionar, um cliente pode se conectar a outro JDBCServer normal.

Comparado com o HA ativo/em espera, o HA de instâncias multi-ativas elimina as seguintes restrições:

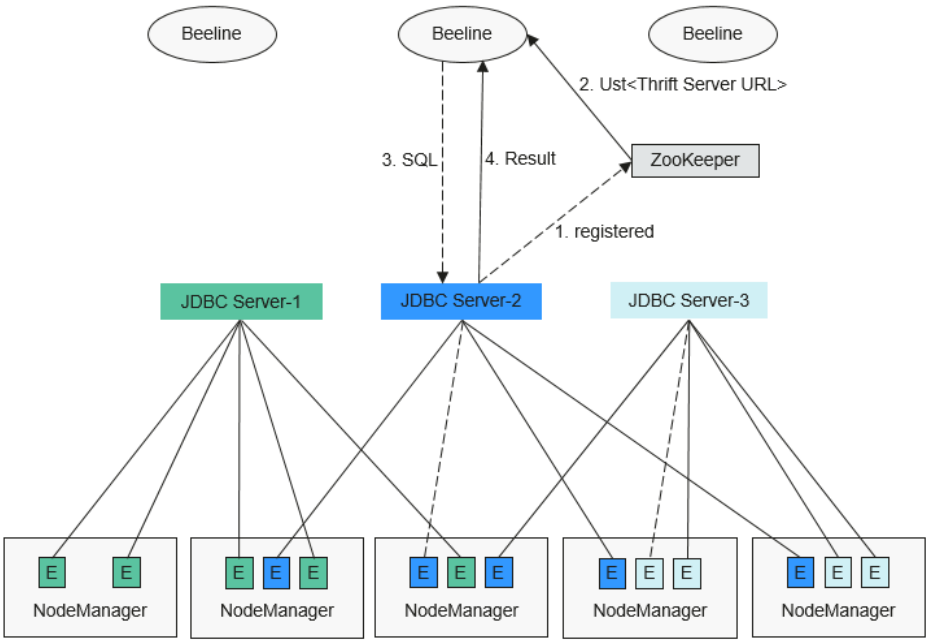
- No HA ativo/em espera, quando a alternância ativa/em espera ocorre, o período indisponível não pode ser controlado pelo JDBCServer, mas depende dos recursos do serviço Yarn.
- No Spark, o Thrift JDBC semelhante ao HiveServer2 fornece serviços e usuários acessam serviços por meio da Beeline e da API de JDBC. Portanto, a capacidade de processamento do cluster de JDBCServer depende da capacidade de ponto único do servidor primário, e a escalabilidade é insuficiente.

O modo HA de instância multi-ativa não só pode impedir a interrupção do serviço causada pela alternância, mas também permite a expansão do cluster para garantir a alta simultaneidade.

Implementação

A figura a seguir mostra o princípio básico do HA da instância multi-ativa do Spark JDBCServer.

Figura 6-114 Spark JDBCServer HA



1. Depois que um JDBCServer é iniciado, ele se registra no ZooKeeper gravando as informações do nó em um diretório especificado. As informações do nó incluem o IP da instância do JDBCServer, o número da porta, a versão e o número de série (as informações de diferentes nós são separadas por vírgulas).

   Um exemplo é fornecido como segue:

   ```
   [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 se conectar ao JDBCServer, o cliente deve especificar o namespace, que é o diretório das instâncias do JDBCServer em ZooKeeper. Durante a conexão, uma instância do JDBCServer é selecionada aleatoriamente no namespace especificado. Para obter detalhes sobre o URL, consulte **Conexão de URL**.
3. Após a conexão ser bem-sucedida, o cliente envia instruções SQL para o JDBCServer.
4. JDBCServer executa instruções SQL recebidas e envia resultados para o cliente.

No modo HA de instâncias multi-ativas, todas as instâncias do JDBCServer são independentes e equivalentes. Quando uma instância do JDBCServer é interrompida durante a atualização, outras instâncias do JDBCServer podem aceitar a solicitação de conexão do cliente.

As seguintes regras devem ser seguidas no HA da instância multi-ativa do Spark JDBCServer:

- Se uma instância JDBCServer sair de forma anormal, nenhuma outra instância assumirá as sessões e serviços em execução nessa instância anormal.
- Quando o processo do JDBCServer é interrompido, os nós correspondentes são excluídos do ZooKeeper.
- O cliente seleciona aleatoriamente o servidor, o que pode resultar em alocação de sessão desigual e, finalmente, resultar em desequilíbrio de carga de instância.

- Depois que a instância entra no modo de manutenção (em que não são aceites novas solicitações de conexão do cliente), os serviços em execução na instância podem falhar quando o descomissionamento atingir o tempo limite.

Conexão de URL

Modo de instância multi-ativa

No modo de instância multi-ativa, o cliente lê o conteúdo do nó do ZooKeeper e se conecta ao JDBCServer. As cadeias de conexão são as seguintes:

- Modo de segurança:
  - Se a autenticação Kinit estiver ativada, o JDBCURL será o seguinte:

    ```
    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 o URL do ZooKeeper. Use vírgulas (,) para separar vários URLs,

      Por exemplo, **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
    - **sparkthriftserver2x** indica o diretório em ZooKeeper onde uma instância aleatória do JDBCServer está conectada ao cliente.

    Por exemplo, quando utiliza o cliente de Beeline para conexão no modo de segurança, execute o seguinte 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*>**;"**
  - Se a autenticação Keytab estiver ativada, o JDBCURL será o seguinte:

    ```
    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 o principal do usuário Kerberos, por exemplo, **test@**<*System domain name*>. <*path_to_keytab*> indica o caminho do arquivo Keytab correspondente a <*principal_name*>, por exemplo, **/opt/auth/test/user.keytab**.
- Modo comum:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
  ```

  Por exemplo, quando utiliza o cliente de Beeline para conexão em modo comum, execute o seguinte 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 instância não multi-ativa

No modo de instância não multi-ativa, um cliente se conecta a um nó do JDBCServer especificado. Comparado com o modo de instância multi-ativa, a cadeia de conexão neste modo de de instância não multi-ativa não contém os parâmetros **serviceDiscoveryMode** e **zooKeeperNamespace** sobre ZooKeeper.

Por exemplo, quando você usa o cliente de Beeline para conectar o JDBCServer no modo de instância não multi-ativa, execute o seguinte 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 o URL do nó JDBCServer especificado.
- **CLIENT_HOME** indica o caminho do cliente.

Exceto o método de conexão, as operações da API de JDBCServer no modo de instância multi-ativa e no modo de instância não-multi-ativa são as mesmas. Spark JDBCServer é outra implementação do HiveServer2 no Hive. Para detalhes sobre como usar o Spark JDBCServer, acesse o site oficial do Hive em **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

6.28.2.2 Spark2x multi-locatário

Conhecimento de fundo

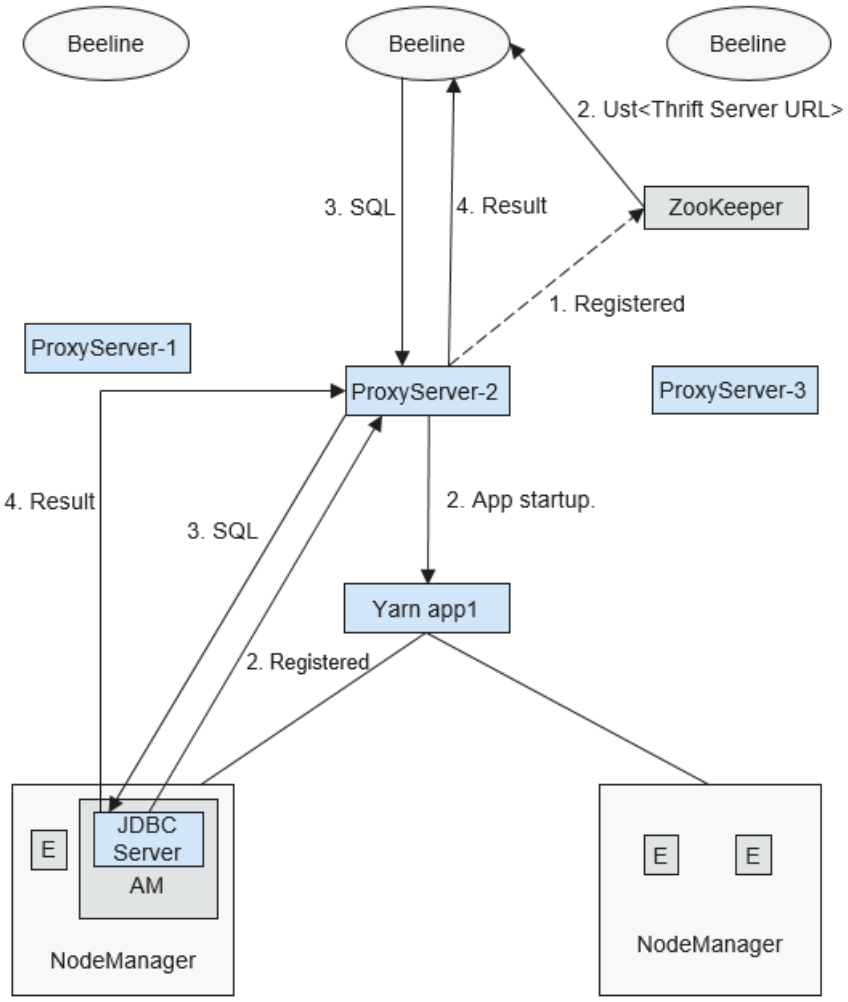
No modo de instância multi-ativa do JDBCServer, o JDBCServer implementa o modo Yarn-client, mas apenas uma fila de recursos Yarn está disponível. Para resolver o problema de limitação de recursos, o modo multi-locatário é introduzido.

No modo multi-locatário, os JDBCServers são vinculados aos locatários. Cada locatário corresponde a um ou mais JDBCServers, e um JDBCServer fornece serviços para apenas um locatário. Locatários diferentes podem ser configurados com filas de Yarn diferentes para implementar o isolamento de recursos. Além disso, o JDBCServer pode ser iniciado dinamicamente conforme necessário para evitar o desperdício de recursos.

Implementação

Figura 6-115 mostra a solução HA do modo multi-locatário.

Figura 6-115 Modo multi-locatário do Spark JDBCServer



1. Quando um ProxyServer é iniciado, ele se registra no ZooKeeper escrevendo as informações do nó em um diretório especificado. As informações do nó incluem o IP da instância, o número da porta, a versão e o número de série (as informações de diferentes nós são separadas por vírgulas).

   NOTA

   No modo multi-locatário, a instância do JDBCServer na página MRS indica ProxyServer, o agente do JDBCServer.

   Um exemplo é fornecido como segue:

   ```
   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 se conectar a ProxyServer, o cliente deve especificar um namespace, que é o diretório da instância do ProxyServer onde você deseja acessar no ZooKeeper. Quando o cliente se conecta a ProxyServer, uma instância em Namespace é selecionada aleatoriamente para conexão. Para obter detalhes sobre o URL, consulte **Conexão de URL**.
3. Depois que o cliente se conecta com sucesso ao ProxyServer, o ProxyServer primeiro verifica se o JDBCServer de um locatário existe. Se sim, o Beeline conecta ao

JDBCServer. Se não, um novo JDBCServer será iniciado no modo Yarn-cluster. Após a inicialização do JDBCServer, o ProxyServer obtém o endereço IP do JDBCServer e estabelece a conexão entre o Beeline e o JDBCServer.

4. O cliente envia instruções SQL para ProxyServer que, em seguida, encaminha instruções para o JDBCServer conectado. JDBCServer retorna os resultados para ProxyServer que, em seguida, retorna os resultados para o cliente.

No modo HA multi-locatário, todas as instâncias do ProxyServer são independentes e equivalentes. Se uma instância for interrompida durante a atualização, outras instâncias poderão aceitar a solicitação de conexão do cliente.

Conexão de URL

Modo multi-locatário

No modo multi-locatário, o cliente lê o conteúdo do nó do ZooKeeper e se conecta ao ProxyServer. As cadeias de conexão são as seguintes:

- Modo de segurança:
  - Se a autenticação Kinit estiver ativada, o URL de cliente será o seguinte:

    ```
    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 o URL do ZooKeeper. Use vírgulas (,) para separar vários URLs,

      Por exemplo, **192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181**.
    - **sparkthriftserver2x** indica o diretório do ZooKeeper, onde uma instância aleatória do JDBCServer está conectada ao cliente.

    Por exemplo, quando utiliza o cliente de Beeline para conexão no modo de segurança, execute o seguinte 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***>;"**
  - Se a autenticação Keytab estiver ativada, o URL será o seguinte:

    ```
    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 o principal do usuário Kerberos, por exemplo, **test@**<*System domain name*>. <*path_to_keytab*> indica o caminho do arquivo Keytab correspondente a <*principal_name*>, por exemplo, **/opt/auth/test/user.keytab**.
- Modo comum:

  ```
  jdbc:hive2://
  <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
  ```


Por exemplo, quando utiliza o cliente de Beeline para conexão em modo comum, execute o seguinte 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 não multi-locatário

No modo não multi-locatário, um cliente se conecta a um nó do JDBCServer especificado. Comparado com o modo de instância multi-ativa, a cadeia de conexão neste modo não multi-locatário não contém os parâmetros **serviceDiscoveryMode** e **zooKeeperNamespace** sobre ZooKeeper.

Por exemplo, quando você usa o cliente de Beeline para conectar o JDBCServer no modo de instância não multi-locatário, execute o seguinte 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 o URL do nó do JDBCServer especificado.
- **CLIENT_HOME** indica o caminho do cliente.

Exceto o método de conexão, outras operações da API do JDBCServer no modo multi-locatário e no modo não multi-locatário são as mesmas. Spark JDBCServer é outra implementação do HiveServer2 no Hive. Para detalhes sobre como usar o Spark JDBCServer, acesse o site oficial do Hive em **https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**.

Especificar um locatário

Geralmente, o cliente enviado por um usuário se conecta ao JDBCServer padrão do locatário ao qual o usuário pertence. Se quiser conectar o cliente ao JDBCServer de um locatário especificado, adicione o parâmetro **--hiveconf mapreduce.job.queuename**.

O comando para conectar o Beeline é o seguinte (**aaa** indica o nome do locatário):

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 Relação entre Spark2x e outros componentes

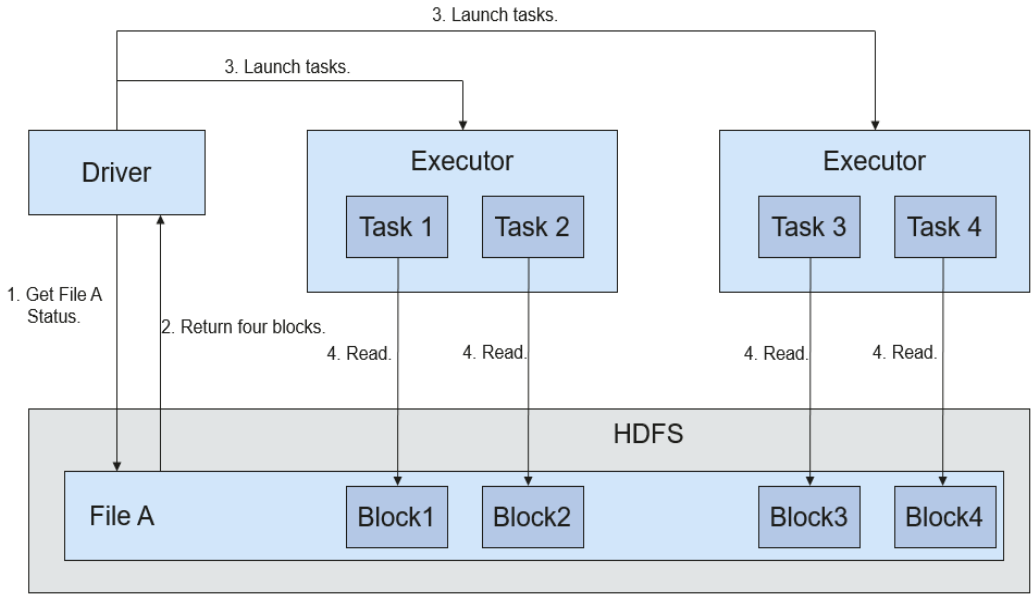
Relação entre Spark e HDFS

Os dados calculados pelo Spark vêm de várias fontes de dados, como arquivos locais e HDFS. A maioria dos dados vem do HDFS, que pode ler dados em grande escala para computação paralela. Após serem computados, os dados podem ser armazenados no HDFS.

O Spark envolve o Driver e o Executor. O Driver agenda tarefas e o Executor executa tarefas.

Figura 6-116 descreve o processo de leitura do arquivo.

Figura 6-116 Processo de leitura de arquivos

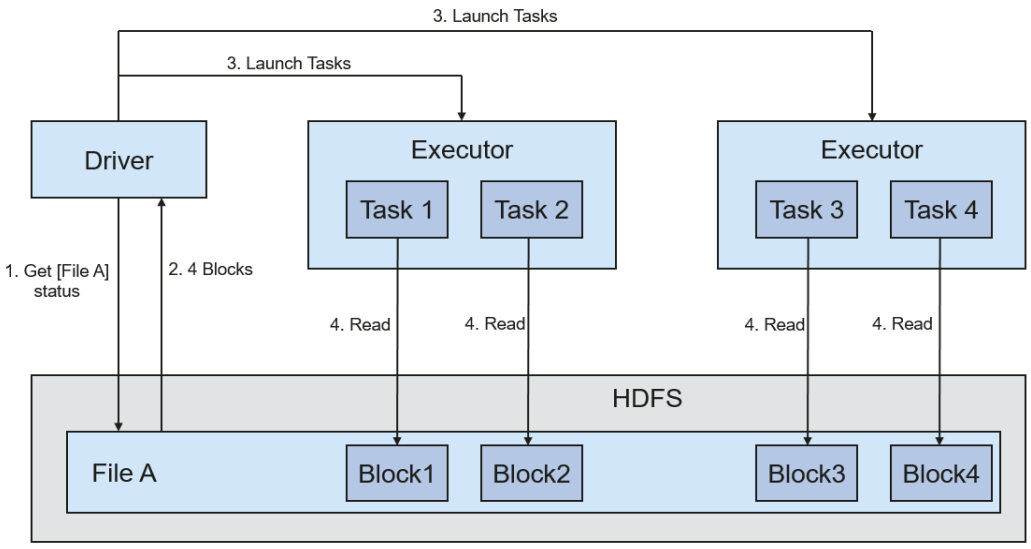


O processo de leitura do arquivo é o seguinte:

1. O Driver se interconecta com o HDFS para obter as informações do Arquivo A.
2. O HDFS retorna as informações de bloco detalhadas sobre esse arquivo.
3. O Driver define um grau paralelo com base na quantidade de dados do bloco e cria várias tarefas para ler os blocos desse arquivo.
4. O Executor executa as tarefas e lê os blocos detalhados como parte do Conjunto de dados distribuído resiliente (RDD).

Figura 6-117 descreve o processo de gravação do arquivo.

Figura 6-117 Processo de gravação de arquivos



O processo de escrita do arquivo é o seguinte:

1. O Driver cria um diretório onde o arquivo deve ser gravado.
2. Com base no status de distribuição do RDD, o número de tarefas relacionadas à gravação de dados é calculado e essas tarefas são enviadas ao Executor.
3. Executor executa essas tarefas e grava os dados do RDD no diretório criado em **1**.

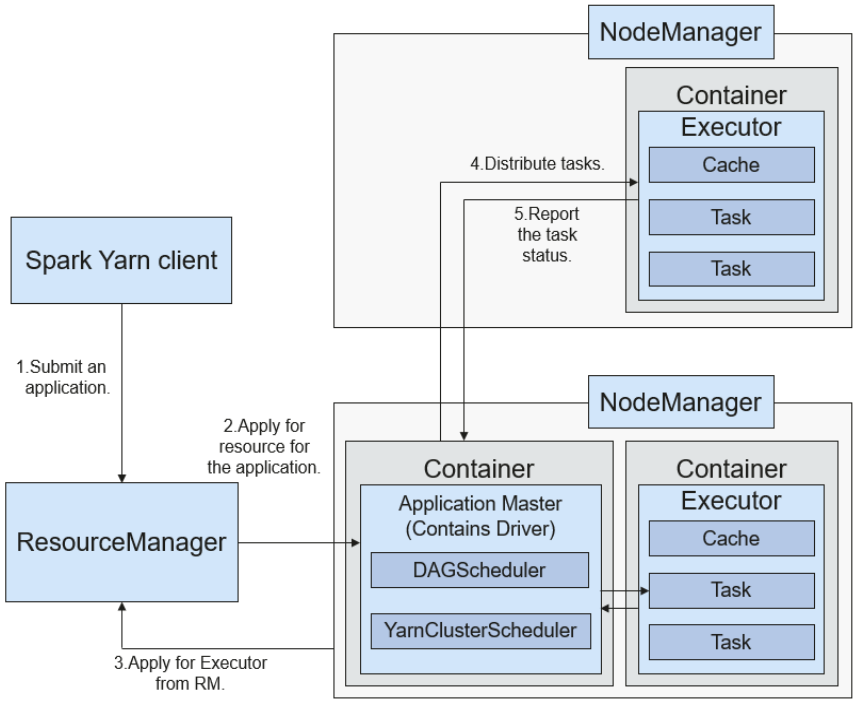
Relação com o Yarn

A computação e o agendamento do Spark podem ser implementados usando o modo Yarn. O Spark desfruta dos recursos de computação fornecidos pelos clusters de Yarn e executa tarefas de forma distribuída. Spark no Yarn tem dois modos: Yarn-cluster e Yarn-client.

- Modo Yarn-cluster

  Figura 6-118 descreve o quadro operacional.

  Figura 6-118 Estrutura de operação de Spark no Yarn-cluster



  Processo de implementação do Spark no Yarn-cluster:

  a. O cliente gera as informações da aplicação e, em seguida, envia as informações para o ResourceManager.
  b. O ResourceManager aloca o primeiro contêiner (ApplicationMaster) para o SparkApplication e inicia o driver no contêiner.
  c. O ApplicationMaster se aplica a recursos do ResourceManager para executar o contêiner.

     O ResourceManager aloca os contêineres para o ApplicationMaster, que se comunica com os NodeManagers relacionados e inicia o executor no contêiner obtido. Depois que o executor é iniciado, ele se registra com os drivers e aplica-se para as tarefas.

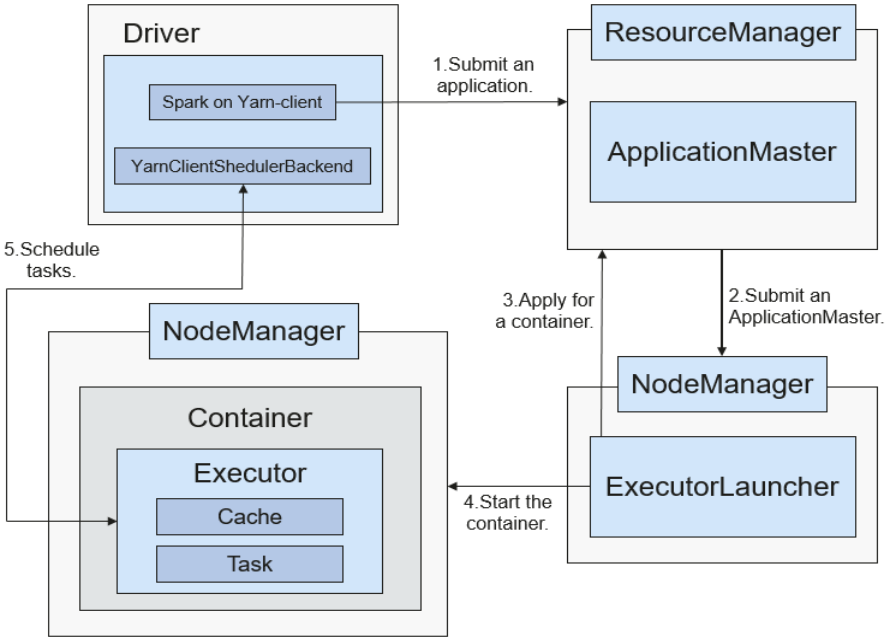
d. Drivers alocam tarefas para os executores.

e. Os executores executam tarefas e relatam o status operacional aos Drivers.

- Modo Yarn-client

  Figura 6-119 descreve o quadro operacional.

Figura 6-119 Estrutura de operação de Spark no Yarn-client



Processo de implementação do Spark no Yarn-client:

NOTA

No modo YARN-client, o Driver é implementado e iniciado no cliente. No modo YARN-client, o cliente de uma versão anterior é incompatível. O modo Yarn-cluster é recomendado.

a. O cliente envia a solicitação da aplicação Spark ao ResourceManager e empacota todas as informações necessárias para iniciar o ApplicationMaster e envia as informações ao ResourceManager. O ResourceManager então retorna os resultados para o cliente. Os resultados incluem informações como ApplicationId, o limite superior e o limite inferior dos recursos disponíveis. Depois de receber a solicitação, o ResourceManager encontra um nó adequado para o ApplicationMaster e o inicia nesse nó. ApplicationMaster é uma função no Yarn e o nome do processo no Spark é ExecutorLauncher.

b. Com base nos requisitos de recursos de cada tarefa, o ApplicationMaster pode solicitar uma série de Contêineres para executar tarefas do ResourceManager.

c. Depois de receber a lista de contêineres recém-alocada (do ResourceManager), o ApplicationMaster envia informações aos NodeManagers relacionados para iniciar os contêineres.

   O ResourceManager aloca os contêineres para o ApplicationMaster, que se comunica com os NodeManagers relacionados e inicia o executor no contêiner obtido. Depois que o executor é iniciado, ele se registra com os drivers e aplica-se para as tarefas.

NOTA

A execução de Contêineres não será suspensa para liberar recursos.

d. Drivers alocam tarefas para os executores. Executores executam tarefas e relatam o status operacional aos Drivers.

6.28.4 Novos recursos de código aberto do Spark2x

Objetivo

Comparado com o Spark 1.5, o Spark2*x* tem alguns novos recursos de código aberto. Os recursos ou conceitos específicos são os seguintes:

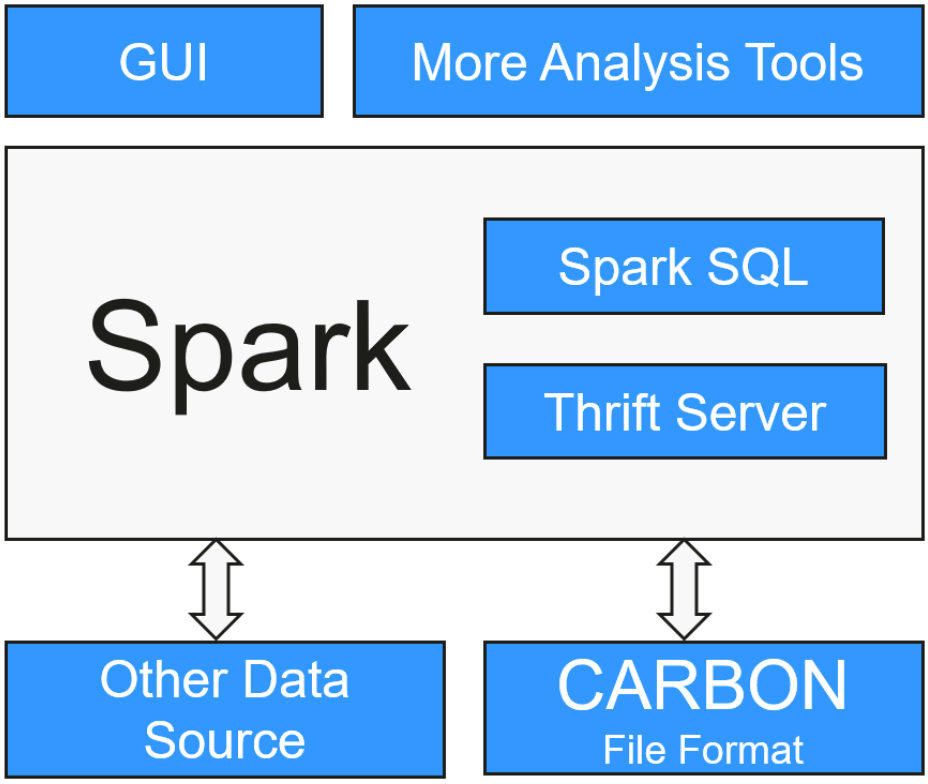
- DataSet: para mais detalhes, consulte **Princípio do SparkSQL e do DataSet**.
- DDL/DML do Spark SQL nativo: para mais detalhes, consulte **Princípio do SparkSQL e do DataSet**.
- SparkSession: para mais detalhes, consulte **Princípio de SparkSession**.
- Streaming estruturado: para mais detalhes, consulte **Princípio do Streaming Estruturado**.
- Otimização de arquivos pequenos
- Otimização do algoritmo agregado
- Otimização de tabelas de Datasource
- Fusão de CBO

6.28.5 Recursos de código aberto aprimorados do Spark2x

6.28.5.1 Visão geral do CarbonData

CarbonData é um novo formato de armazenamento de dados nativo do Apache Hadoop. CarbonData permite consultas interativas mais rápidas em PetaBytes de dados usando técnicas avançadas de armazenamento colunar, índice, compactação e codificação para melhorar a eficiência da computação. Além disso, o CarbonData também é um mecanismo de análise de alto desempenho que integra fontes de dados ao Spark.

Figura 6-120 Arquitetura básica do CarbonData



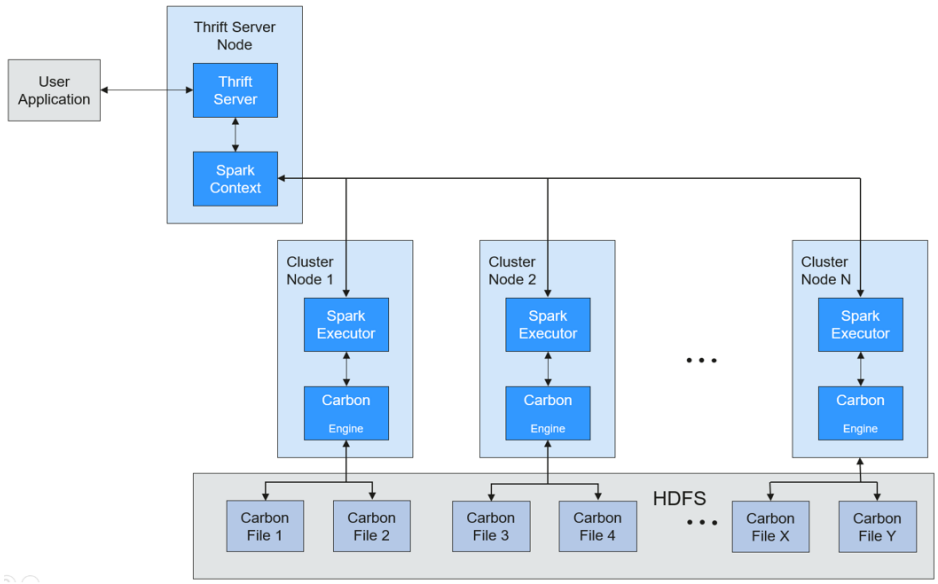
O propósito de usar CarbonData é fornecer resposta rápida a consultas ad hoc de Big Data. Essencialmente, o CarbonData é um mecanismo de Processamento Analítico On-line (OLAP), que armazena dados usando tabelas semelhantes às do Sistema de Gerenciamento de Banco de Dados Relacional (RDBMS). Você pode importar mais de 10 TB de dados para tabelas criadas no formato CarbonData e o CarbonData organiza e armazena automaticamente os dados usando os índices multidimensionais compactados. Depois que os dados são carregados em CarbonData, o CarbonData responde a consultas ad hoc em segundos.

O CarbonData integra fontes de dados ao ecossistema Spark e você pode consultar e analisar os dados usando o Spark SQL. Você também pode usar a ferramenta de terceiros JDBCServer fornecida pelo Spark para se conectar ao SparkSQL.

Topologia do CarbonData

O CarbonData é executado como uma fonte de dados dentro do Spark. Portanto, o CarbonData não inicia nenhum processo adicional em nós em clusters. O motor CarbonData funciona dentro do executor do Spark.

Figura 6-121 Topologia do CarbonData



Os dados armazenados na CarbonData Table são divididos em vários arquivos de dados do CarbonData. Sempre que os dados são consultados, o CarbonData Engine lê e filtra os conjuntos de dados. CarbonData Engine é executado como parte do processo Executor do Spark e é responsável por manipular um subconjunto de blocos de arquivos de dados.

Os dados da tabela são armazenados no HDFS. Os nós no mesmo cluster do Spark podem ser usados como nós de dados do HDFS.

Recursos do CarbonData

- SQL: CarbonData é compatível com o Spark SQL e suporta operações de consulta SQL executadas no Spark SQL.
- Definição de conjunto de dados de tabela simples: CarbonData permite definir e criar conjuntos de dados usando instruções de Linguagem de Definição de Dados (DDL). CarbonData DDL é flexível e fácil de usar, e pode definir tabelas complexas.
- Fácil gerenciamento de dados: CarbonData fornece várias funções de gerenciamento de dados para carregamento e manutenção de dados. O CarbonData suporta o carregamento em massa de dados históricos e o carregamento incremental de novos dados. Os dados carregados podem ser excluídos com base no tempo de carregamento e uma operação de carregamento específica pode ser desfeita.
- O formato de arquivo CarbonData é um armazenamento colunar no HDFS. Esse formato tem muitos novos recursos de armazenamento de arquivos baseados em colunas, como divisão de tabelas e compactação de dados. CarbonData tem as seguintes características:
  - Armazena dados junto com o índice: acelera significativamente o desempenho da consulta e reduz as varreduras de I/O e os recursos da CPU, quando há filtros na consulta. O índice de CarbonData consiste em vários níveis de índices. Uma estrutura de processamento pode aproveitar esse índice para reduzir a tarefa que precisa ser agendada e processada e também pode executar a varredura de pular em uma unidade de grãos mais fina (chamada blocklet) na varredura do lado da tarefa em vez de verificar o arquivo inteiro.
  - Dados codificados operáveis: com o suporte a esquemas eficientes de compressão e codificação global, CarbonData pode consultar dados compactados/codificados. Os

dados podem ser convertidos pouco antes de retornar os resultados para os usuários, o que é chamado de materialização tardia.

- Suporta vários casos de uso com um único formato de dados: como consulta interativa no estilo OLAP, Acesso Sequencial (grande varredura) e Acesso Aleatório (varredura estreita).

Principais tecnologias e vantagens do CarbonData

- Resposta rápida da consulta: CarbonData apresenta consultas de alto desempenho. A velocidade de consulta do CarbonData é 10 vezes maior que a do Spark SQL. Ele usa formatos de dados dedicados e aplica várias tecnologias de índice, código de dicionário global e várias otimizações push-down, fornecendo resposta rápida a consultas de dados em nível de TB.
- Compressão de dados eficiente: CarbonData comprime dados combinando os algoritmos de compressão leves e pesados. Isso economiza significativamente de 60% a 80% de espaço de armazenamento de dados e o custo de armazenamento de hardware.

Servidor de cache de índice do CarbonData

Para resolver a pressão e os problemas trazidos pelo aumento do volume de dados para o driver, um servidor de cache de índice independente é introduzido para separar o índice do lado do aplicação Spark da consulta Carbon. Todo o conteúdo do índice é gerenciado pelo servidor de cache de índice. As aplicações Spark obtêm os dados de índice necessários no modo RPC. Desta forma, uma grande quantidade de memória no lado do serviço é liberada para que os serviços não sejam afetados pela escala de cluster e o desempenho ou as funções não sejam afetados.

6.28.5.2 Otimização da consulta SQL de dados de várias fontes

Cenário

As empresas geralmente armazenam dados massivos, como de vários bancos de dados e armazéns, para gerenciamento e coleta de informações. No entanto, fontes de dados diversificadas, estruturas de conjuntos de dados híbridos e armazenamento de dados dispersos reduzem a eficiência da consulta.

O Spark de código aberto só suporta pushdown de filtro simples durante a consulta de dados de várias fontes. O desempenho do mecanismo SQL está deteriorado devido a uma grande quantidade de transmissão de dados desnecessária. A função pushdown é aprimorada para que **aggregate**, **projection** complexa e **predicate** complexa possam ser enviados para fontes de dados, reduzindo a transmissão de dados desnecessária e melhorando o desempenho da consulta.

Somente a origem de dados JDBC suporta pushdown de operações de consulta, como **aggregate**, **projection**, **predicate**, **aggregate over inner join** e **aggregate over union all**. Todas as operações de pushdown podem ser ativadas com base em suas necessidades.

Tabela 6-25 Consulta aprimorada de consulta entre fontes

Módulo	Antes do aprimoramento	Após o aprimoramento
aggregate	O pushdown da **aggregate** não é suportado.	● funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. Exemplo: selecione count(*) da tabela ● Expressões internas de funções de agregação são suportadas. Exemplo: selecione sum(a+b) da tabela ● O cálculo de funções de agregação é suportado. Exemplo: selecione avg(a) + max(b) da tabela ● Pushdown de **having** é suportado. Exemplo: selecione sum(a) da tabela onde a>0 group by b tendo sum(a)>10 ● O pushdown de algumas funções é suportado. Pushdown de linhas em matemática, tempo e funções de cadeia, como **abs()**, **month()** e **length()** são suportados. Além das funções internas anteriores, você pode executar o comando **SET** para adicionar funções suportadas pelas fontes de dados. Exemplo: selecione sum(abs(a)) da tabela ● Pushdown de **limit** e **order by** após **aggregate** é suportado. No entanto, o pushdown não é suportado no Oracle, porque o Oracle não suporta **limit**. Exemplo: selecione sum(a) da tabela onde a>0 group by b order by sum(a) limit 5
projection	Apenas pushdown de **projection** simples é suportado. Exemplo: selecione a, b da tabela	● Expressões complexas podem ser empurradas para baixo. Exemplo: selecione (a+b)*c da tabela ● Algumas funções podem ser empurradas para baixo. Para obter detalhes, consulte a descrição abaixo da tabela. Exemplo: selecione length(a)+abs(b) da tabela ● Pushdown do **limit** e **order by** após a **projection** é suportada. Exemplo: selecione a, b+c da tabela order by um limit 3

Módulo	Antes do aprimoramento	Após o aprimoramento
predicate	Somente a filtragem simples com o nome da coluna à esquerda do operador e os valores à direita é suportada. Exemplo: selecione * da tabela em que a>0 ou b em ("aaa", "bbb")	● O pushdown de expressões complexas é suportado. Exemplo: selecione * da tabela em que a +b>c*d ou a/c em (1, 2, 3) ● Algumas funções podem ser empurradas para baixo. Para obter detalhes, consulte a descrição abaixo da tabela. Exemplo: selecione * da tabela onde length(a)>5
aggregate over inner join	Os dados relacionados das duas tabelas devem ser carregados no Spark. A operação de junção deve ser executada antes da operação **aggregate**.	Há suporte para as seguintes funções: ● funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. ● Todas as operações **aggregate** podem ser realizadas em uma mesma tabela. As operações **group by** podem ser executadas em uma ou duas tabelas e somente a junção interna é suportada. Os seguintes cenários não são suportados: ● **aggregate** não pode ser empurrado para baixo a partir das tabelas de junção esquerda e direita. ● **aggregate** contém operações, por exemplo, sum(a+b). ● Operações **aggregate**, por exemplo, sum(a)+min(b).
aggregate over union all	Os dados relacionados das duas tabelas devem ser carregados no Spark. **union** deve ser realizada antes de **aggregate**.	Cenários suportados: funções de agregação incluindo **sum**, **avg**, **max**, **min** e **count** são suportadas. Cenários não suportados: ● **aggregate** contém operações, por exemplo, sum(a+b). ● Operações **aggregate**, por exemplo, sum(a)+min(b).

Precauções

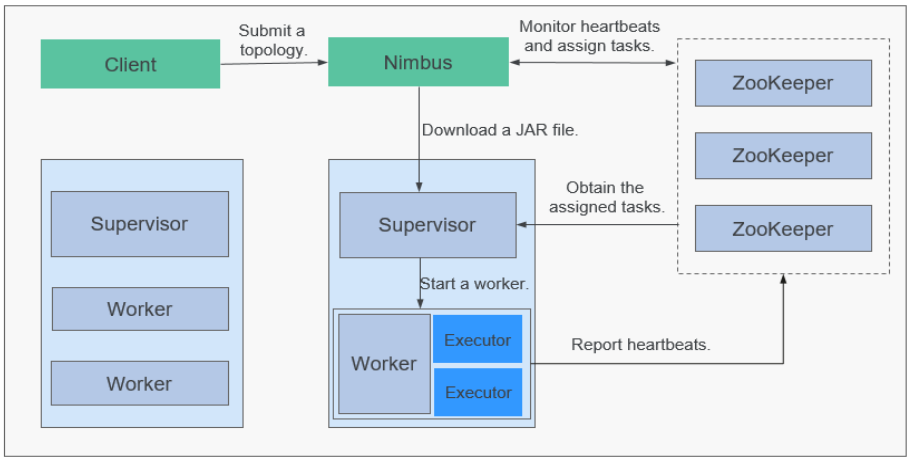
- Se a fonte de dados externa for Hive, a operação de consulta não poderá ser executada em tabelas externas criadas pelo Spark.
- Somente fontes de dados MySQL e MPPDB são suportadas.

6.29 Storm

6.29.1 Princípios básicos do Storm

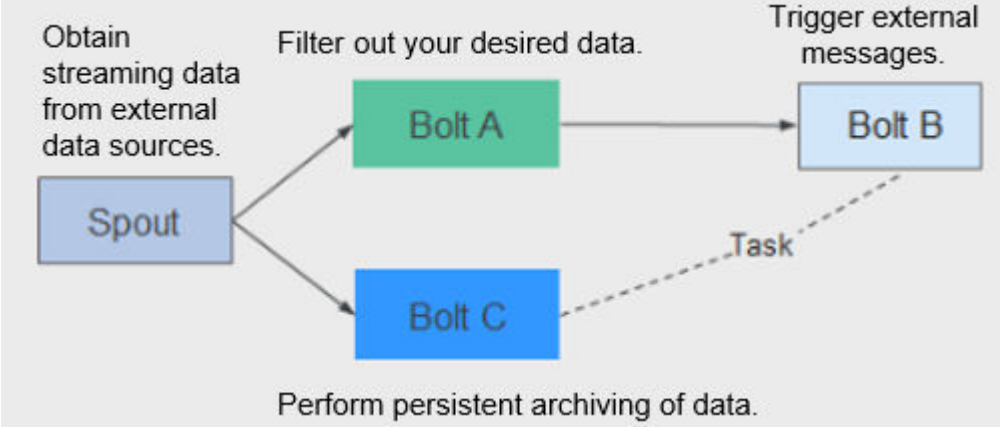
O Apache Storm é um sistema de processamento de dados de fluxo em tempo real distribuído, confiável e tolerante a falhas. No Storm, uma estrutura de dados em forma de gráfico chamada topologia precisa ser projetada primeiro para computação em tempo real. A topologia será submetida a um cluster. Em seguida, um nó principal no cluster distribui códigos e atribui tarefas aos nós de trabalho. Uma topologia contém duas funções: spout e bolt. Um spout envia mensagens e envia fluxos de dados em tuplas. Um bolt converte os fluxos de dados e executa operações de computação e filtragem. O bolt pode enviar aleatoriamente dados para outros bolts. Tuplas enviadas por um spout são matrizes imutáveis e mapeiam para pares de chave-valor fixos.

Figura 6-122 Arquitetura do sistema do Storm



A lógica de processamento do serviço é encapsulada na topologia do Storm. Uma topologia é um conjunto de componentes de spout (fontes de dados) e bolt (processamento lógico) que são conectados usando agrupamentos de fluxo no modo DAG. Todos os componentes (spout e bolt) em uma topologia estão trabalhando em paralelo. Em uma topologia, você pode especificar o paralelismo para cada nó. Em seguida, o Storm aloca tarefas no cluster para computação para melhorar os recursos de processamento do sistema.

Figura 6-123 Topologia



O Storm é aplicável à análise em tempo real, computação contínua e extração, transformação e carga distribuídas (ETL). Ele possui as seguintes vantagens:

- Aplicações amplas
- Alta escalabilidade
- Zero perda de dados
- Alta tolerância a falhas
- Fácil de construir e controlar
- Compatibilidade com diferentes linguagens

O Storm é uma plataforma de computação e fornece a Linguagem de Consulta Contínua (CQL) na camada de serviço para facilitar a implementação do serviço. CQL possui os seguintes recursos:

- Facilidade de uso: a sintaxe CQL é semelhante à sintaxe SQL. Os usuários que possuem conhecimentos básicos da SQL podem facilmente aprender CQL e usá-la para desenvolver serviços.
- Funções ricas: além das expressões básicas fornecidas pela SQL, a CQL fornece funções, como janelas, filtragem e configuração de simultaneidade, para processamento de fluxo.
- Fácil de escalar: CQL fornece uma API de extensão para suportar cenários de serviços cada vez mais complexos. Os usuários podem personalizar a entrada, a saída, a serialização e a desserialização para atender a requisitos específicos de serviço.
- Fácil de depurar: CQL fornece uma explicação detalhada dos códigos de erro, facilitando os usuários a corrigir falhas.

Para detalhes sobre a arquitetura e os princípios do Storm, veja **https://storm.apache.org/**.

Princípio

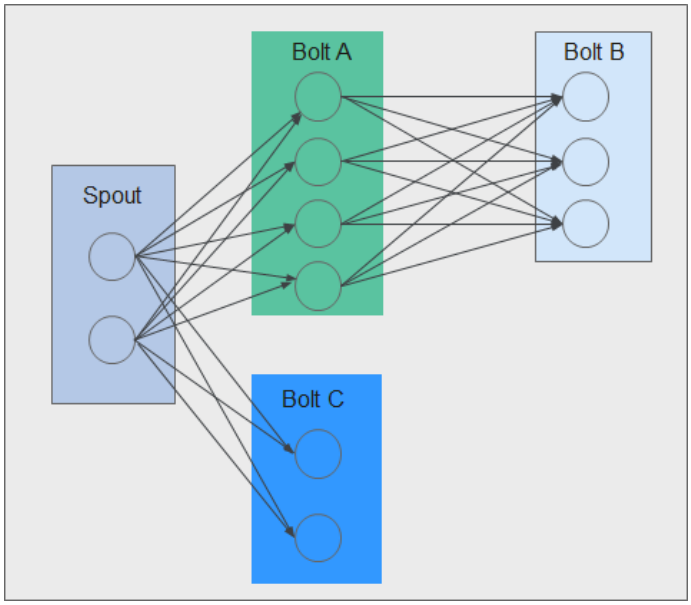
- **Conceitos básicos**

Tabela 6-26 Conceitos

Conceito	Descrição
Tupla	Uma tupla é um par chave-valor invariável usado para transferir dados. Tuplas são criadas e processadas de forma distribuída.
Fluxo	Um fluxo é uma sequência ilimitada de tuples.
Topologia	Uma topologia é um aplicativo em tempo real executado na plataforma Storm. É um Gráfico Acíclico Dirigido (DAG) composto por componentes. Uma topologia pode ser executada simultaneamente em várias máquinas. Cada máquina executa uma parte do DAG. Uma topologia é semelhante a um job de MapReduce. A diferença é que a topologia é um programa residente. Uma vez iniciada, a topologia não pode parar a menos que seja terminada manualmente.
Spout	Um spout é a fonte de tuples. Por exemplo, um spout pode ler dados de uma fila de mensagens, banco de dados, sistema de arquivos ou conexão TCP e convertê-los como tuplas, que são processadas pelo próximo componente.
Bolt	Em uma Topologia, um bolt é um componente que recebe dados e executa uma lógica específica, como filtragem ou conversão de tuplas, junção ou agregação de fluxos e execução de estatísticas e persistência de resultados.
Worker	Um Worker é um processamento físico em estado de execução em uma Topologia. Cada Worker é um processo JVM. Cada Topologia pode ser executada por vários Worker. Cada Worker executa um subconjunto lógico da Topologia.
Task	Uma tarefa é um thread de spout ou bolt de um Worker.
Agrupamento de fluxo	Um agrupamento de fluxo especifica as políticas de envio de tuples. Ele instrui o bolt subsequente como receber tuplas. As políticas suportadas incluem Shuffle Grouping, Fields Grouping, All Grouping, Global Grouping, Non Grouping e Directed Grouping.

Figura 6-124 mostra uma Topologia (DAG) que consiste em um Spout e um Bolt. Na figura, um retângulo indica um Spout ou Bolt, o nó em cada retângulo indica tarefas e as linhas entre tarefas indicam fluxos.

Figura 6-124 Topologia



- **Confiabilidade**

  O Storm fornece três níveis de confiabilidade de dados:

  - No máximo uma vez: os dados processados podem ser perdidos, mas não podem ser processados repetidamente. Este nível de confiabilidade oferece a maior taxa de transferência.
  - No mínimo uma vez: os dados podem ser processados repetidamente para garantir uma transmissão confiável de dados. Se uma resposta não for recebida dentro do tempo especificado, o Spout reenvia os dados para a Bolts para processamento. Esse nível de confiabilidade pode afetar ligeiramente o desempenho do sistema.
  - Exatamente uma vez: os dados são transmitidos com sucesso sem perda ou processamento de redundância. Esse nível de confiabilidade oferece o pior desempenho.

  Selecione o nível de confiabilidade com base nos requisitos de serviço. Por exemplo, para os serviços que exigem alta confiabilidade de dados, use Exatamente uma vez para garantir que os dados sejam processados apenas uma vez. Para os serviços insensíveis à perda de dados, use outros níveis para melhorar o desempenho do sistema.

- **Tolerância a falhas**

  Storm é um sistema tolerante a falhas que oferece alta disponibilidade. **Tabela 6-27** descreve a tolerância a falhas dos componentes do Storm.

Tabela 6-27 Tolerância a falhas

Cenário	Descrição
Nimbus falhou	Nimbus é fail-fast e sem estado. Se o Nimbus ativo estiver com defeito, o Nimbus em espera assumirá os serviços imediatamente e fornecerá serviços externos.

Cenário	Descrição
O supervisor falhou	Supervisor é um daemon de fundo dos Workers. É fail-fast e sem estado. Se um Supervisor estiver com defeito, os Workers em execução no nó não serão afetados, mas não poderão receber novas tarefas. O OMS pode detectar a falha do Supervisor e reiniciar os processos.
Worker falhou	Se um Worker estiver com defeito, o Supervisor no Worker o reiniciará novamente. Se a reinicialização falhar várias vezes, o Nimbus reatribuirá tarefas a outros nós.
Nó falhou	Se um nó estiver com defeito, todas as tarefas que estão sendo processadas pelo nó expiram e o Nimbus atribuirá as tarefas a outro nó para processamento.

Recursos de código aberto

- Computação distribuída em tempo real

  Em um cluster do Storm, cada máquina suporta a execução de vários processos de trabalho e cada processo de trabalho pode criar vários threads. Cada thread pode executar várias tarefas. Uma tarefa indica o processamento simultâneo de dados.
- Alta tolerância a falhas

  Durante o processamento de mensagens, se um nó ou um processo estiver defeituoso, a unidade de processamento de mensagens pode ser reimplementada.
- Mensagens confiáveis

  Métodos de processamento de dados, incluindo At-Least Once, At-Most Once e Exactly Once são suportados.
- Mecanismo de segurança

  O Storm fornece autenticação baseada em Kerberos e mecanismos de autorização conectáveis, suporta SSL Storm UI e Log Viewer UI e suporta integração de segurança com outros componentes da plataforma de Big Data (como ZooKeeper e HDFS).
- Definição e implementação de topologia flexível

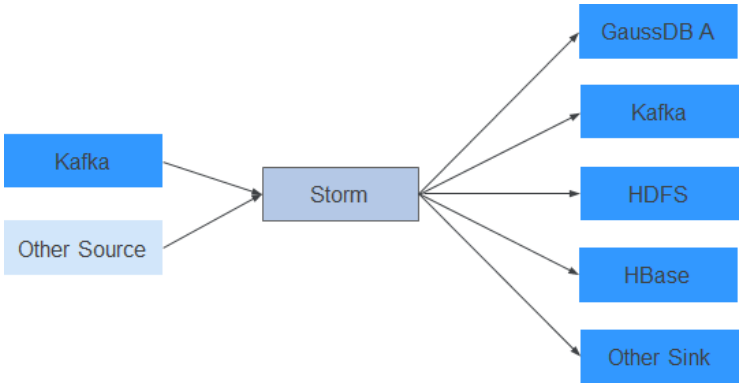
  A estrutura Flux é usada para definir e implementar topologias de serviços. Se o DAG de serviço for alterado, os usuários só precisarão modificar a linguagem específica de domínio YAML (DSL), mas não precisarão recompilar ou empacotar o código de serviço.
- Integração com componentes externos

  O Storm oferece suporte à integração com vários componentes externos, como Kafka, HDFS, HBase, Redis e JDBC/RDBMS, implementando serviços que envolvem várias fontes de dados.

6.29.2 Relação entre Storm e outros componentes

O Storm fornece uma estrutura de computação distribuída em tempo real. Ele pode obter mensagens em tempo real de fontes de dados (como Kafka e conexão TCP), executar computação em tempo real de alta taxa de transferência e baixa latência em uma plataforma em tempo real e exportar resultados para filas de mensagens ou implementar persistência de dados. **Figura 6-125** mostra a relação entre Storm e outros componentes.

Figura 6-125 Relação com outros componentes



Relação entre Storm e Streaming

Tanto o Storm quanto o Streaming usam o Apache Storm kernel de código aberto. No entanto, a versão do kernel usada pelo Storm é a 1.2.1, enquanto a usada pelo Streaming é a 0.10.0. Streaming é usado para herdar serviços de transição em cenários de atualização. Por exemplo, se o Streaming tiver sido implantado em uma versão anterior e os serviços estiverem em execução, o Streaming ainda poderá ser usado após a atualização. Storm é recomendado em um novo cluster.

Storm 1.2.1 tem as seguintes novas funcionalidades:

- **Cache distribuído**: fornece recursos externos (configurações) necessários para compartilhar e atualizar a topologia usando ferramentas CLI. Você não precisa reempacotar e re-implementar a topologia.
- **API de janela de Streaming nativa**: fornece APIs baseadas em janela.
- **Agendador de recursos**: adicionou o plug-in do agendador de recursos. Ao definir uma topologia, você pode especificar o máximo de recursos disponíveis e atribuir cotas de recursos aos usuários, para gerenciar os recursos de topologia dos usuários.
- **Gestão de estado**: fornece a API de Bolt com o mecanismo de ponto de verificação. Quando um evento falha, Storm gerencia automaticamente o status de Bolt e restaura o evento.
- **Amostragem e depuração de mensagens**: na interface do usuário do Storm, você pode ativar ou desativar a depuração em nível de topologia ou componente para enviar mensagens de fluxo para logs especificados com base na taxa de amostragem.
- **Análise dinâmica de Worker**: na IU do Storm, você pode coletar logs de jstack e heap do processo de Worker e reiniciar o processo de Worker.
- **Ajuste dinâmico de logs de topologia**: você pode alterar dinamicamente os logs de topologia em execução na CLI ou na IU do Storm.
- **Desempenho melhorado**: em comparação com as versões anteriores, o desempenho de Storm é bastante melhorado. Embora o desempenho da topologia esteja intimamente relacionado ao cenário de caso de uso e à dependência de serviços externos, o desempenho é três vezes maior na maioria dos cenários.

6.29.3 Recursos de código aberto aprimorados do Storm

- CQL

  Continuous Query Language (CQL) é uma linguagem semelhante a SQL usada para processamento de fluxo em tempo real. Comparado com a SQL, a CQL introduziu o

conceito de janela (sequenciamento do tempo), que permite que os dados sejam armazenados e processados na memória. A saída de CQL é os resultados de computação de fluxos de dados em um momento específico. O uso de CQL acelera o desenvolvimento de serviços, permite que as tarefas sejam facilmente enviadas à plataforma Storm para processamento em tempo real, facilita a saída de resultados e permite que as tarefas sejam encerradas no momento apropriado.

- Alta disponibilidade

  Nimbus HA garante o processamento contínuo de serviços, como a adição de topologias e gerenciamento, mesmo que um Nimbus esteja com defeito, melhorando a disponibilidade do cluster.

6.30 Tez

Tez é a mais recente estrutura de computação de código aberto do Apache que suporta jobs de Grafo Direcionado Acíclico (DAG). Ele pode converter vários jobs dependentes em um job, melhorando significativamente o desempenho dos jobs de DAG.

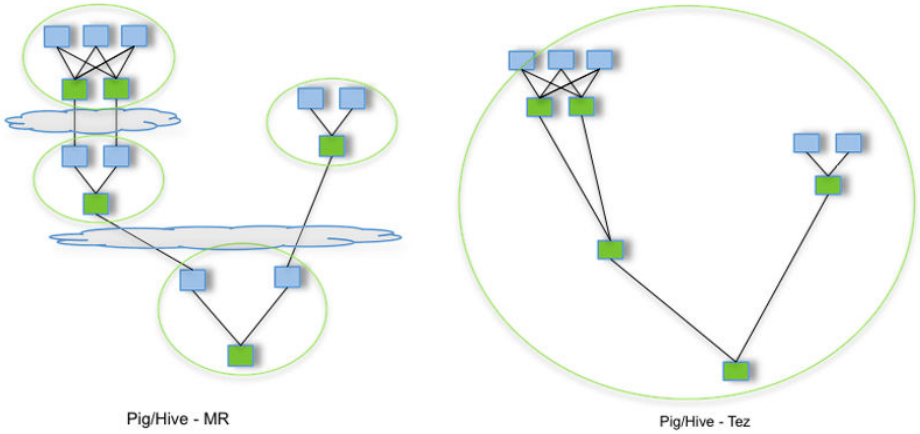
O MRS usa o Tez como mecanismo de execução padrão do Hive. Tez supera notavelmente o mecanismo de computação do MapReduce original em termos de eficiência de execução.

Para detalhes sobre Tez, veja **https://tez.apache.org/**.

Relação entre Tez e MapReduce

Tez usa um DAG para organizar tarefas de MapReduce. No DAG, um nó é um RDD e uma borda indica uma operação no RDD. A ideia central é dividir ainda mais as tarefas Map e Reduce. Uma tarefa Map é dividida nas tarefas Input-Processor-Sort-Merge-Output, e a tarefa Reduce é dividida nas tarefas Input-Shuffle-Sort-Merge-Process-output. Tez reagrupa de forma flexível várias pequenas tarefas para formar um grande job de DAG.

Figura 6-126 Processos para enviar tarefas usando o Hive no MapReduce e o Hive no Tez



Uma tarefa de Hive no MapReduce contém várias tarefas do MapReduce. Cada tarefa armazena resultados intermediários no HDFS. O redutor na etapa anterior fornece dados para o mapeador na próxima etapa. Uma tarefa de Hive on Tez pode concluir o mesmo processo de processamento em apenas uma tarefa, e o HDFS não precisa ser acessado entre as tarefas.

Relação entre Tez e Yarn

Tez é uma estrutura de computação executando no Yarn. O ambiente de tempo de execução consiste em ResourceManager e ApplicationMaster do Yarn. O ResourceManager é um novo sistema de gerenciamento de recursos, e o ApplicationMaster é responsável por cortar dados de jobs do MapReduce, atribuir tarefas, solicitar recursos, agendar tarefas e tolerar falhas. Além disso, TezUI depende do TimelineServer fornecido pelo Yarn para exibir o processo de execução das tarefas do Tez.

6.31 YARN

6.31.1 Princípios básicos do YARN

A comunidade Apache apresenta a estrutura de gerenciamento de recursos unificado do YARN para compartilhar clusters do Hadoop, melhorar sua escalabilidade e confiabilidade e eliminar um gargalo de desempenho de JobTracker na estrutura inicial do MapReduce.

A ideia fundamental do YARN é dividir as duas principais funcionalidades do JobTracker, gerenciamento de recursos e agendamento/monitoramento de jobs, em daemons separados. A ideia é ter uma ResourceManager global (RM) e uma ApplicationMaster por aplicação (AM).

NOTA

Uma aplicação é um job único no sentido clássico de jobs do MapReduce ou um Gráfico Acíclico Dirigido (DAG) de jobs.

Arquitetura

ResourceManager é a essência da estrutura em camadas do YARN. Essa entidade controla um cluster inteiro e gerencia a alocação de aplicações aos recursos de computação subjacentes. O ResourceManager aloca cuidadosamente vários recursos (computação, memória, largura de banda e assim por diante) para NodeManagers subjacentes (agentes por nó do YARN). O ResourceManager também trabalha com o ApplicationMasters para alocar recursos e trabalha com o NodeManagers para iniciar e monitorar suas aplicações subjacentes. Nesse contexto, o ApplicationMaster assumiu parte do papel do TaskTracker anterior, e o ResourceManager assumiu o papel do JobTracker.

O ApplicationMaster gerencia cada instância de uma aplicação em execução no YARN. O ApplicationMaster negocia recursos do ResourceManager e trabalha com os NodeManagers para monitorar a execução do contêiner e o uso de recursos (alocação de recursos de CPU e memória).

O NodeManager gerencia cada nó em um cluster do YARN. O NodeManager fornece serviços por nó em um cluster, desde supervisionar o gerenciamento de um contêiner durante seu ciclo de vida até monitorar recursos e rastrear a saúde de seus nós. O MRv1 gerencia a execução das tarefas Map e Reduce por meio de slots, enquanto o NodeManager gerencia contêiners abstratos, que representam recursos por nó disponíveis para um aplicativo específico.

Figura 6-127 Arquitetura

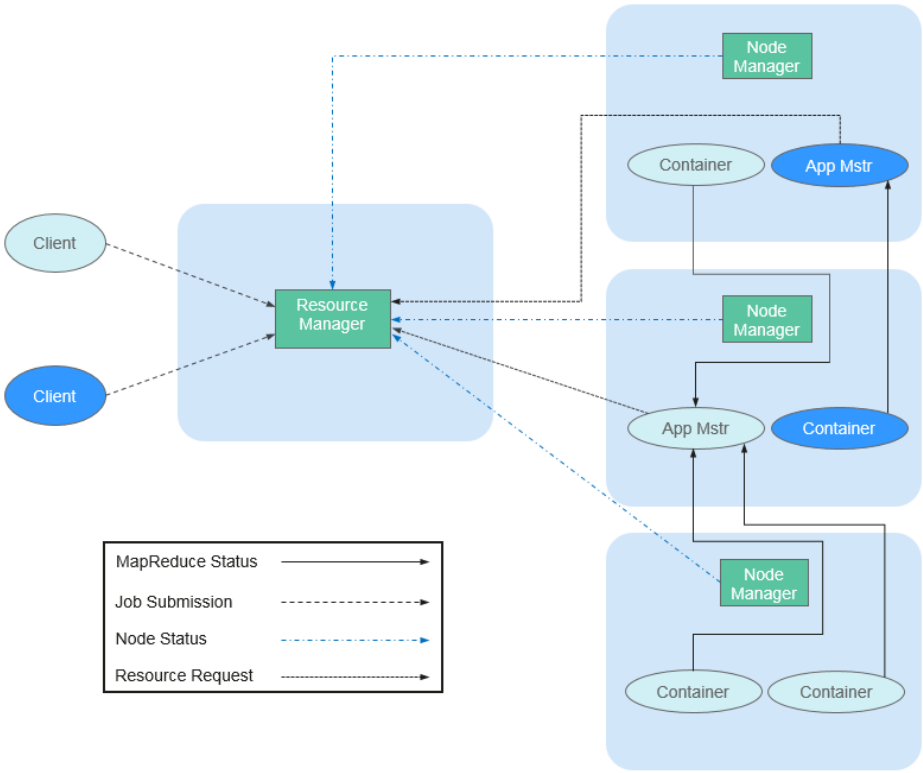


Tabela 6-28 descreve os componentes mostrados em **Figura 6-127**.

Tabela 6-28 Descrição da arquitetura

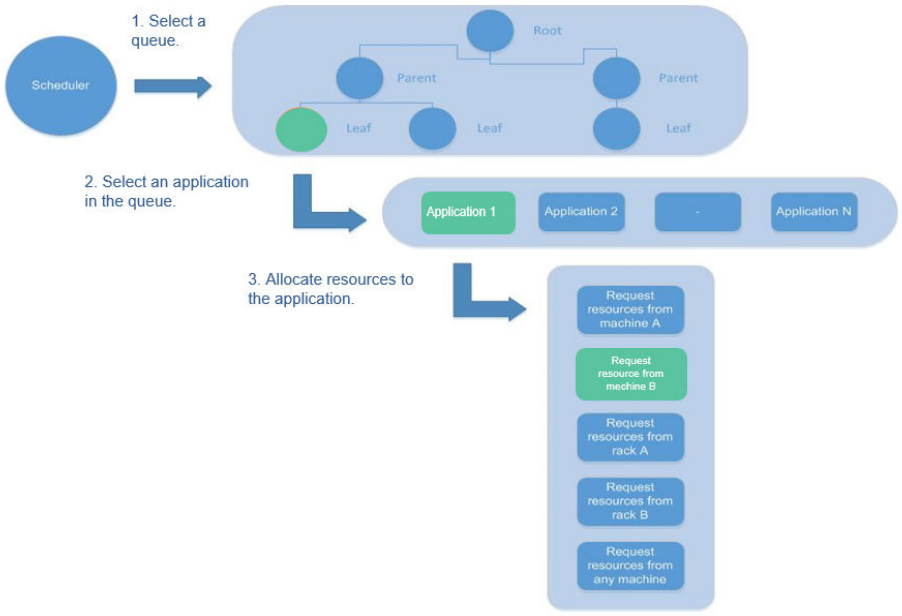
Nome	Descrição
Client	Cliente de uma aplicação do YARN. Você pode enviar uma tarefa para a ResourceManager e consultar o status operacional de uma aplicação usando o cliente.
ResourceManager(RM)	O RM gerencia e aloca centralmente todos os recursos no cluster. Ele recebe informações de cada nó (NodeManager) e aloca recursos para aplicações com base nos recursos coletados de acordo com uma política especificada.
NodeManager(NM)	NM é o agente em cada nó do YARN. Gerencia o nó de computação no cluster do Hadoop, estabelece comunicação com o ResourceManger e monitora o ciclo de vida dos contêineres, monitora o uso de recursos como memória e CPU de cada contêiner, rastreia o status do nó e gerencia logs e serviços auxiliares usados por diferentes aplicações.
Application Master (AM)	AM (App Mstr na figura acima) é responsável por todas as tarefas através do ciclo de vida de em uma aplicação. As tarefas incluem o seguinte: negociar com um agendador de RM para obter um recurso; alocar ainda mais os recursos obtidos para tarefas internas (alocação secundária de recursos); comunicar com o NM para iniciar ou parar tarefas; monitorar o status de execução de todas as tarefas; e solicitar recursos para tarefas novamente para reiniciar as tarefas quando as tarefas não forem executadas.

Nome	Descrição
Container	Uma abstração de recursos no YARN. Ele encapsula recursos multidimensionais (incluindo apenas memória e CPU) em um determinado nó. Quando o ApplicationMaster solicita recursos do ResourceManager, o ResourceManager retorna recursos para o ApplicationMaster em um contêiner. O YARN aloca um contêiner para cada tarefa e a tarefa só pode usar os recursos encapsulados no contêiner.

No YARN, os agendadores de recursos organizam recursos por meio de filas hierárquicas. Isso garante que os recursos sejam alocados e compartilhados entre filas, melhorando assim o uso de recursos de cluster. O modelo de alocação de recursos principal do Superior Scheduler é o mesmo do Capacity Scheduler, conforme mostrado na figura a seguir.

Um agendador mantém as informações da fila. Você pode enviar aplicações para uma ou mais filas. Durante cada pulsação do NM, o agendador seleciona uma fila de acordo com uma regra de agendamento específica, seleciona uma aplicação na fila e aloca recursos para a aplicação. Se os recursos não forem alocados para a aplicação devido ao limite de alguns parâmetros, o agendador selecionará outra aplicação. Após a seleção, o agendador processa a solicitação de recurso desta aplicação. O agendador dá prioridade às solicitações de recursos locais primeiro, e depois para recursos no mesmo rack e, finalmente, para recursos de qualquer máquina.

Figura 6-128 Modelo de alocação de recursos



Princípio

A nova estrutura de MapReduce do Hadoop é chamada de MRv2 ou YARN. YARN é composto por ResourceManager, ApplicationMaster e NodeManager.

- O ResourceManager é um gerenciador de recursos global que gerencia e aloca recursos no sistema. O ResourceManager é composto pelo Agendador e pelo Gerenciador de aplicações.

  - O Agendador aloca recursos do sistema para todas as aplicações em execução com base nas restrições, como capacidade e fila (por exemplo, aloca uma certa quantidade de recursos para uma fila e executa um número específico de jobs). Ele aloca recursos com base na demanda de aplicações, com o contêiner sendo usado como a unidade de alocação de recursos. Funcionando como uma unidade dinâmica de alocação de recursos, o Contêiner encapsula recursos de memória, CPU, disco e rede, limitando assim o recurso consumido por cada tarefa. Além disso, o Agendador é um componente plugável. Você pode projetar novos agendadores conforme necessário. O YARN fornece vários agendadores diretamente disponíveis, como o Fair Scheduler e o Capacity Scheduler.
  - O Gerenciador de aplicações gerencia todas as aplicações no sistema e envolve o envio de aplicações, negociação com agendadores sobre recursos, ativação e monitoramento do ApplicationMaster e reinicialização do ApplicationMaster em caso de falha de inicialização.
- O NodeManager é o gerenciador de recursos e tarefas de cada nó. Por um lado, o NodeManager informa periodicamente o uso de recursos do nó local e o status de execução de cada Contêiner para ResourceManager. Por outro lado, o NodeManager recebe e processa solicitações de ApplicationMaster para iniciar ou interromper Contêineres.
- O ApplicationMaster é responsável por todas as tarefas durante o ciclo de vida de um aplicativo, esses canais incluem o seguinte:
  - Negociar com o agendador RM para obter recursos.
  - Atribuir recursos a componentes internos (alocação secundária de recursos).
  - Comunicar-se com o NodeManager para iniciar ou interromper tarefas.
  - Monitorar o status de execução de todas as tarefas e aplique recursos novamente para tarefas quando as tarefas não forem executadas para reiniciar as tarefas.

Princípio do Capacity Scheduler

O Capacity Scheduler é um agendador multiusuário. Ele aloca recursos por fila e define os recursos mínimos/máximos que podem ser usados para cada fila. Além disso, o limite superior de uso de recursos é definido para cada usuário para evitar abuso de recursos. Os recursos restantes de uma fila podem ser compartilhados temporariamente com outras filas.

O Capacity Scheduler suporta várias filas. Ele configura uma certa quantidade de recursos para cada fila e adota a política de agendamento de enfileiramento primeiro a entrar primeiro a sair (FIFO). Para impedir que os aplicativos de um usuário usem exclusivamente os recursos em uma fila, o Capacity Scheduler define um limite para o número de recursos usados por jobs enviados por um usuário. Durante a programação, o Capacity Scheduler calcula primeiro o número de recursos necessários para cada fila e seleciona a fila que requer menos recursos. Em seguida, ele aloca recursos com base na prioridade do job e no tempo em que os jobs são enviados, bem como no limite de recursos e memória. O Capacity Scheduler oferece suporte aos seguintes recursos:

- Capacidade garantida: como administrador do cluster de MRS, você pode definir os limites inferior e superior de uso de recursos para cada fila. Todas as aplicações enviadas a essa fila compartilham os recursos.
- Alta flexibilidade: temporariamente, os recursos restantes de uma fila podem ser compartilhados com outras filas. No entanto, esses recursos devem ser liberados em caso de envio de novo pedido para a fila. Essa alocação flexível de recursos ajuda a melhorar notavelmente o uso de recursos.

- Múltiplos locatários: vários usuários podem compartilhar um cluster e várias aplicações podem ser executadas simultaneamente. Para evitar o uso exclusivo de recursos por uma única aplicação, usuário ou fila, o administrador do cluster do MRS pode adicionar várias restrições (por exemplo, limite em tarefas simultâneas de uma única aplicação).
- Proteção assegurada: uma lista de ACL é fornecida para cada fila para limitar estritamente o acesso do usuário. Você pode especificar os usuários que podem exibir o status do aplicativo ou controlar os aplicativos. Além disso, o administrador de cluster do MRS pode especificar um administrador de fila e um administrador de sistema de cluster.
- Atualização dinâmica dos arquivos de configuração: os administradores de cluster do MRS podem modificar dinamicamente os parâmetros de configuração para gerenciar clusters on-line.

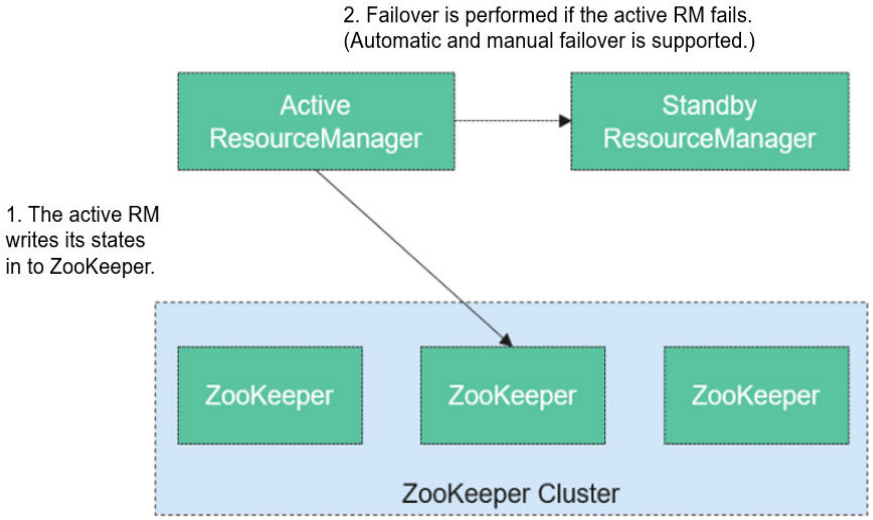
Cada fila no Capacity Scheduler pode limitar o uso de recursos. No entanto, o uso de recursos de uma fila determina sua prioridade quando os recursos são alocados às filas, indicando que as filas com menor capacidade são competitivas. Se a taxa de transferência de um cluster for grande, o agendamento de atraso permite que uma aplicação desista do agendamento entre máquinas ou racks e solicite o agendamento local.

6.31.2 Solução HA do YARN

Princípios e solução de implementação de HA

O ResourceManager no YARN gerencia recursos e agenda tarefas no cluster. Em versões anteriores ao Hadoop 2.4, SPOFs podem ocorrer em ResourceManager no cluster do YARN. A solução HA do YARN usa nós redundantes de ResourceManager para enfrentar desafios de confiabilidade de serviço e tolerância a falhas.

Figura 6-129 Arquitetura de HA do ResourceManager



HA de ResourceManager é conseguido usando nós ResourceManager ativo-em espera, como mostrado em **Figura 6-129**. Semelhante à solução HA do HDFS, o HA do ResourceManager permite que apenas um nó do ResourceManager esteja no estado ativo a qualquer momento. Quando o ResourceManager ativo falha, a alternância ativo-em espera pode ser disparado automaticamente ou manualmente.

Quando a função de failover automático não estiver desativada, após o cluster do YARN, os administradores de cluster do MRS precisarão executar o comando ***yarn rmadmin*** para

alternar manualmente um dos nós do ResourceManager para o estado ativo. Em um evento de manutenção planejado ou uma falha, espera-se que primeiro rebaixem o ResourceManager ativo para o estado de espera e o ResourceManager de espera para o estado ativo.

Quando o failover automático é ativado, um ActiveStandbyElector interno baseado no ZooKeeper é usado para decidir qual nó do ResourceManager deve ser o ativo. Quando o ResourceManager ativo é defeituoso, outro nó do ResourceManager é automaticamente selecionado para ser o ativo para assumir o nó defeituoso.

Quando os nós ResourceManager do cluster são implantados no modo HA, a configuração **yarn-site.xml** usada pelos clientes precisa listar todos os nós do ResourceManager. O cliente (incluindo ApplicationMaster e NodeManager) procura o ResourceManager ativo no modo de sondagem. Ou seja, o cliente precisa fornecer o mecanismo de tolerância a falhas. Se o ResourceManager ativo não puder ser conectado com, o cliente procurará continuamente por um novo no modo de sondagem.

Depois que o nó do ResourceManager em espera se torna ativo, as aplicações de camada superior podem recuperar seu status quando a falha ocorre. Para obter detalhes, consulte **Reinicialização do ResourceManager**. Quando a Reinicialização do ResourceManager está habilitada, o nó do ResourceManager reiniciado carrega as informações do nó do ResourceManager ativo anterior e assume as informações de status do contêiner em todos os nós do NodeManager para continuar o serviço em execução. Desta forma, as informações de status podem ser salvas através da execução periódica de operações de ponto de verificação, evitando a perda de dados. Certifique-se de que os nós do ResourceManager ativo e em espera possam acessar as informações de status. Atualmente, são fornecidos três métodos para compartilhar informações de status por sistema de arquivos (FileSystemRMStateStore), banco de dados LevelDB (LeveldbRMStateStore) e ZooKeeper (ZKRMStateStore). Entre eles, apenas ZKRMStateStore suporta o mecanismo de Fencing. Por padrão, o Hadoop usa o ZKRMStateStore.

Para obter mais informações sobre a solução HA do YARN, visite o seguinte site:

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 Relação entre YARN e outros componentes

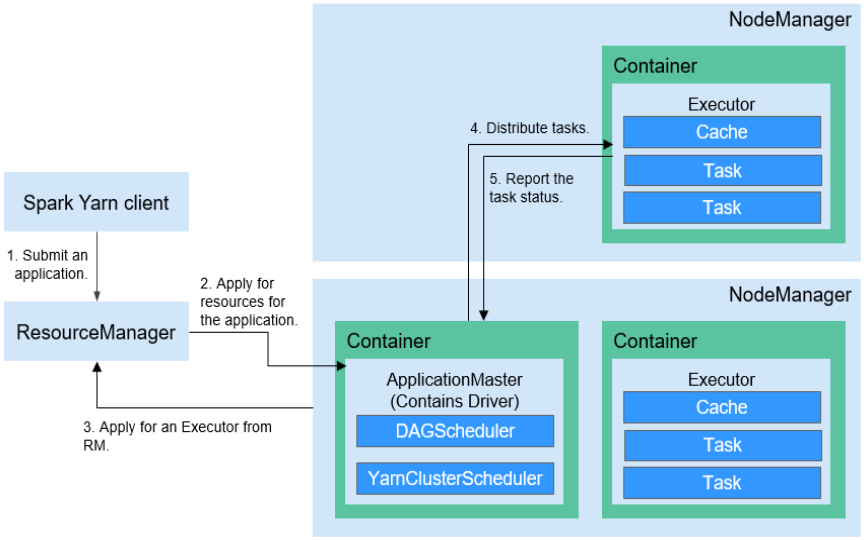
Relação entre YARN e Spark

A computação e o agendamento do Spark podem ser implementados usando o modo YARN. O Spark aproveita os recursos de computação fornecidos pelos clusters de YARN e executa tarefas de maneira distribuída. Spark no YARN tem dois modos: YARN-cluster e YARN-cliente.

- Modo YARN Cluster

  Figura 6-130 descreve o quadro operacional.

Figura 6-130 Estrutura de operação do Spark no YARN-cluster



Processo de implementação do Spark on YARN-cluster:

a. O cliente gera as informações da aplicação e, em seguida, envia as informações para o ResourceManager.

b. O ResourceManager aloca o primeiro contêiner (ApplicationMaster) para o SparkApplication e inicia o driver no contêiner.

c. O ApplicationMaster se aplica a recursos do ResourceManager para executar o contêiner.

   O ResourceManager aloca os contêineres para o ApplicationMaster, que se comunica com os NodeManagers relacionados e inicia o executor no contêiner obtido. Depois que o executor é iniciado, ele se registra com os drivers e aplica-se para as tarefas.

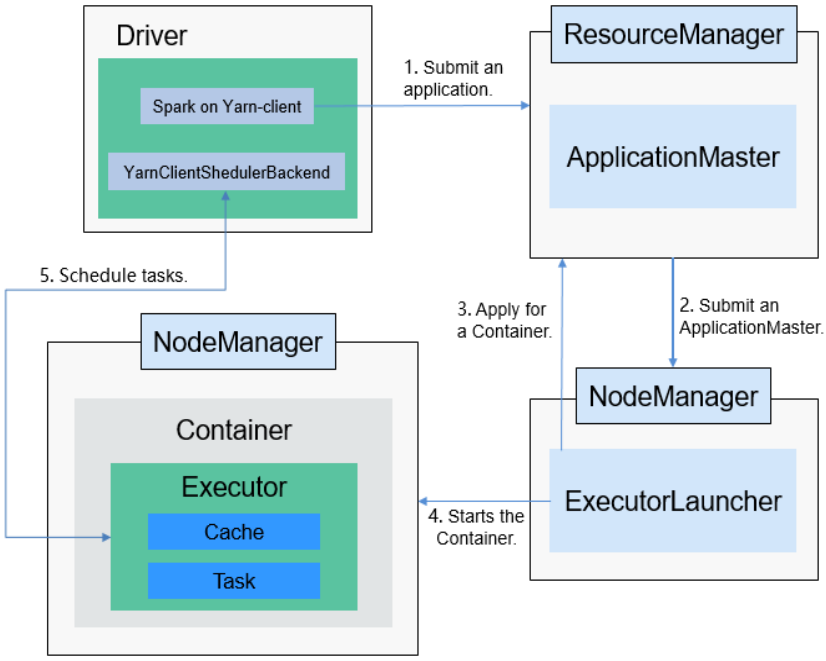
d. Drivers alocam tarefas para os executores.

e. Os executores executam tarefas e relatam o status operacional aos Drivers.

- Modo YARN Client

  Figura 6-131 descreve o quadro operacional.

Figura 6-131 Estrutura de operação do Spark on YARN-client



Processo de implementação do Spark no YARN-client:

NOTA

No modo YARN-client, o driver é implementado e iniciado no cliente. No modo YARN-client, o cliente de uma versão anterior é incompatível. Você é aconselhado a usar o modo YARN-cluster.

a. O cliente envia a solicitação da aplicação de Spark para ResourceManager e, em seguida, o ResourceManager retorna os resultados. Os resultados incluem informações como ID da aplicação e o máximo e mínimo de recursos disponíveis. O cliente empacota todas as informações necessárias para iniciar o ApplicationMaster e envia as informações para ResourceManager.

b. Depois de receber a solicitação, o ResourceManager encontra um nó adequado para o ApplicationMaster e o inicia nesse nó. ApplicationMaster é uma função no YARN e o nome do processo no Spark é ExecutorLauncher.

c. Com base nos requisitos de recursos de cada tarefa, o ApplicationMaster pode solicitar uma série de contêineres para executar tarefas do ResourceManager.

d. Depois de receber a lista de contêineres recém-alocada (do ResourceManager), o ApplicationMaster envia informações aos NodeManagers relacionados para iniciar os contêineres.

   O ResourceManager aloca os contêineres para o ApplicationMaster, que se comunica com os NodeManagers relacionados e inicia o executor no contêiner obtido. Depois que o executor é iniciado, ele se registra com os drivers e aplica-se para as tarefas.

   NOTA

   Os contêineres em execução não são suspensos e os recursos não são liberados.

e. Drivers alocam tarefas para os executores. Os executores executam tarefas e relatam o status operacional aos Drivers.

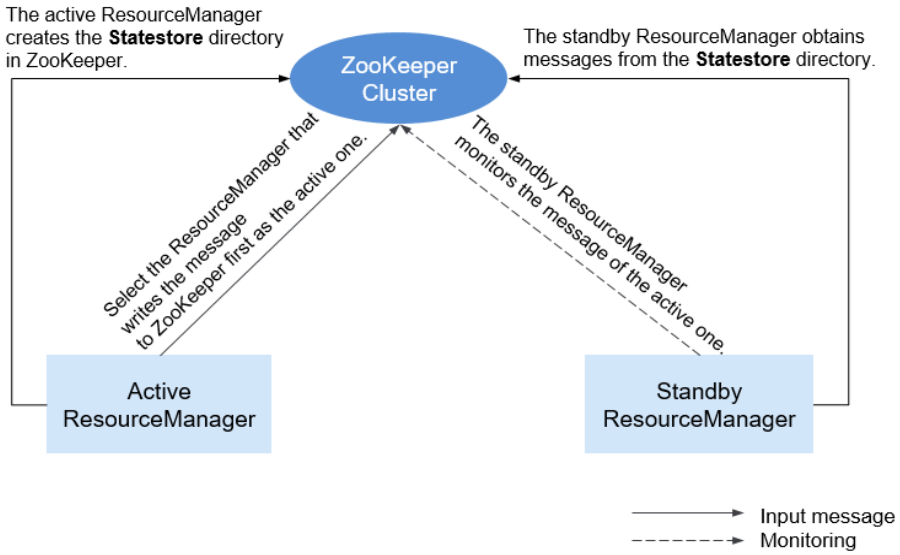
Relação entre YARN e MapReduce

MapReduce é uma estrutura de computação em execução no YARN, que é usado para processamento em lote. MRv1 é implementado com base no MapReduce no Hadoop 1.0, que é composto por modelos de programação (novas e antigas APIs de programação), ambiente de execução (JobTracker e TaskTracker) e mecanismo de processamento de dados (MapTask e ReduceTask). Essa estrutura ainda é fraca em escalabilidade, tolerância a falhas (SPOF JobTracker) e compatibilidade com várias estruturas. (Atualmente, apenas a estrutura de computação MapReduce é suportada.) MRv2 é implementado com base no MapReduce no Hadoop 2.0. O código-fonte reutiliza os modelos de programação MRv1 e a implementação do mecanismo de processamento de dados, e o ambiente em execução é composto por ResourceManager e ApplicationMaster. O ResourceManager é um novo sistema de gerenciamento de recursos, e o ApplicationMaster é responsável por cortar dados de job do MapReduce, atribuir tarefas, solicitar recursos, agendar tarefas e tolerar falhas.

Relação entre YARN e ZooKeeper

Figura 6-132 mostra a relação entre ZooKeeper e YARN.

Figura 6-132 Relação entre ZooKeeper e YARN



1. Quando o sistema é iniciado, o ResourceManager tenta gravar informações de estado em ZooKeeper. O ResourceManager que primeiro grava informações de estado no ZooKeeper é selecionado como o ResourceManager ativo e outros são ResourceManagers em espera. Os ResourceManagers em espera monitoram periodicamente as informações eleitorais ativas do ResourceManager no ZooKeeper.
2. O ResourceManager ativo cria o diretório **Statestore** no ZooKeeper para armazenar informações da aplicação. Se o ResourceManager ativo estiver com defeito, o ResourceManager em espera obterá informações da aplicação do diretório **Statestore** e restaurará os dados.

Relação entre YARN e Tez

As informações do job Hive no Tez requerem a capacidade do TimeLine Server do YARN para que as tarefas do Hive possam exibir o status atual e histórico das aplicações, facilitando o armazenamento e a recuperação.

6.31.4 Recursos de código aberto aprimorados do Yarn

Agendamento de tarefas com base em prioridades

No mecanismo de agendamento de recursos Yarn nativo, se todos os recursos do cluster Hadoop forem ocupados por jobs do MapReduce enviados anteriormente, os jobs enviados posteriormente serão mantidos em estado pendente até que todos os jobs em execução sejam executados e os recursos sejam liberados.

O cluster do MRS fornece o mecanismo de agendamento de prioridade de tarefa. Com esse recurso, você pode definir trabalhos de diferentes prioridades. Trabalhos de alta prioridade podem antecipar recursos liberados de trabalhos de baixa prioridade, embora os trabalhos de alta prioridade sejam enviados posteriormente. Os trabalhos de baixa prioridade que não são iniciados serão suspensos, a menos que esses trabalhos de alta prioridade sejam concluídos e os recursos sejam liberados, então eles podem ser iniciados adequadamente.

Esse recurso permite que os serviços controlem os trabalhos de computação de maneira mais flexível, obtendo assim maior utilização de recursos de cluster.

NOTA

A reutilização de contêiner está em conflito com o agendamento de prioridade de tarefa. Se a reutilização de contêiner estiver habilitada, os recursos estão sendo ocupados e o agendamento de prioridade de tarefa não entrará em vigor.

Controle de permissão de Yarn

O mecanismo de permissão do Hadoop Yarn é implementado por meio de ACLs. O seguinte descreve como conceder diferentes controles de permissão para diferentes usuários:

- ACL de administração

  Um administrador de O&M é especificado para o cluster de YARN. A ACL de administração é determinada por **yarn.admin.acl**. O administrador de O&M do cluster pode acessar a IU da Web do ResourceManager e operar nós do NodeManager, filas e NodeLabel, **mas não pode enviar tarefas**.

- Fila de ACL

  Para facilitar o gerenciamento de usuários no cluster, os usuários ou grupos de usuários são divididos em várias filas às quais cada usuário e grupo de usuários pertence. Cada fila contém permissões para enviar e gerenciar aplicações (por exemplo, encerrar qualquer aplicação).

Funções de código aberto:

atualmente, o Yarn suporta as seguintes funções para usuários:

- Administrador de O&M do cluster
- Administrador de fila
- Usuário comum

No entanto, as APIs (como a IU da Web, a API REST e a API Java) fornecidos pelo Yarn não oferecem suporte ao controle de permissão específico da função. Portanto, todos os usuários têm permissão para acessar as informações do aplicação e do cluster, que não atendem aos requisitos de isolamento no cenário multi-locatário.

Esta é uma função aprimorada.

No modo de segurança, o gerenciamento de permissões é aprimorado para as APIs, como IU da Web, API REST e API Java fornecidas pelo Yarn. O controle de permissão pode ser realizado com base nas funções do usuário.

As permissões baseadas em funções são as seguintes:

- Administrador do Cluster O&M: executa operações de gerenciamento no cluster Yarn, como acessar a IU da Web do ResourceManager, atualizar filas, definir NodeLabel e executar alternância ativa/em espera.
- Administrador de fila: tem permissão para modificar e visualizar filas gerenciadas pelo cluster do Yarn.
- Usuário comum: tem permissão para modificar e exibir aplicativos auto-submetidos no cluster do Yarn.

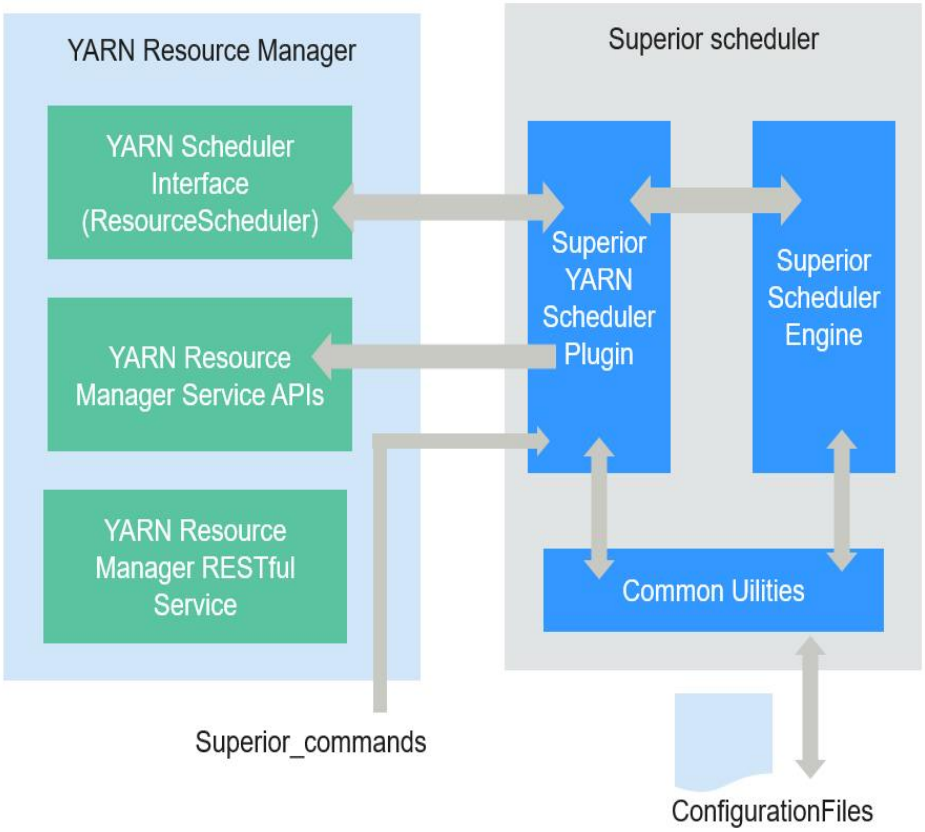
Princípio do Superior Scheduler (Autodesenvolvido)

O Superior Scheduler é um mecanismo de agendamento projetado para o sistema de gerenciamento de recursos distribuídos de Hadoop YARN. É um programador de alto desempenho e de nível empresarial projetado para pools de recursos convergentes e requisitos de serviço multi locatário.

Superior Scheduler atinge todas as funções dos agendadores de código aberto, do Fair Scheduler e do Capacity Scheduler. Em comparação com os agendadores de código aberto, o Superior Scheduler é aprimorado na política de agendamento de recursos multi-locatário da empresa, isolamento de recursos e compartilhamento entre usuários em um locatário, desempenho de agendamento, uso de recursos do sistema e escalabilidade de cluster. O Superior Scheduler foi projetado para substituir os agendadores de código aberto.

Semelhante ao Fair Scheduler e ao Capacity Scheduler, o Superior Scheduler segue a API do plug-in do Yarn para interagir com o Yarn ResourceManager para oferecer funcionalidades de agendamento de recursos. **Figura 6-133** mostra o diagrama geral do sistema.

Figura 6-133 Arquitetura interna do Superior Scheduler



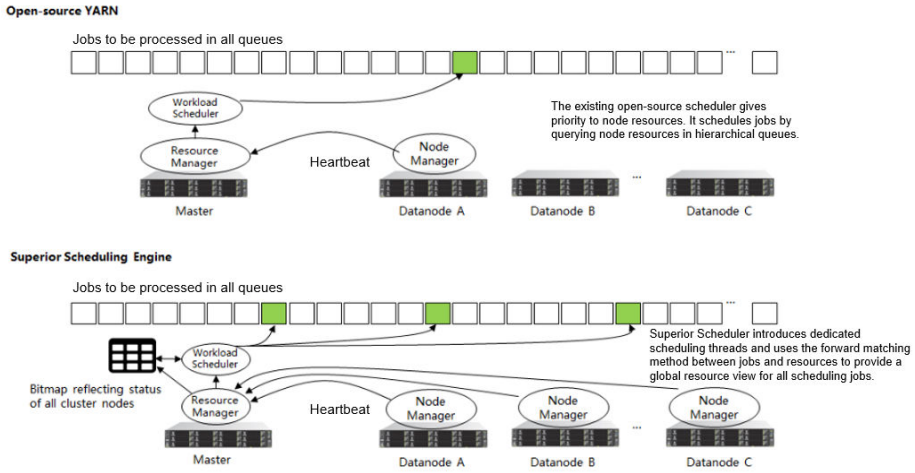
Em **Figura 6-133**, o Superior Scheduler consiste nos seguintes módulos:

- Superior Scheduler Engine é um mecanismo de agendamento de alto desempenho com ricas políticas de agendamento.
- Superior Yarn Scheduler Plugin funciona como uma ponte entre o Yarn ResourceManager e o Superior Scheduler Engine e interage com o Yarn ResourceManager.

  O princípio de agendamento dos programadores de código aberto é que os recursos correspondam aos jobs com base nos batimentos cardíacos dos nós de computação. Especificamente, cada nó de computação envia periodicamente mensagens para ResourceManager de Yarn para notificar o status do nó e inicia o agendador para atribuir tarefas ao próprio nó. Neste mecanismo de agendamento, o período de agendamento depende do batimento cardíaco. Se a escala de cluster aumentar, pode ocorrer gargalo na escalabilidade do sistema e no desempenho de agendamento. Além disso, como os recursos correspondem aos trabalhos, a precisão de agendamento de um agendador de código aberto é limitada. Por exemplo, a afinidade de dados é aleatória e o sistema não suporta políticas de agendamento baseadas em carga. O agendador pode não fazer a melhor escolha devido à falta da exibição de recursos global ao selecionar tarefas.

  O Superior Scheduler adota vários mecanismos de agendamento. Existem tópicos de agendamento dedicados no Superior Scheduler, separando batimentos cardíacos com agendamento e evitando tempestades de batimentos cardíacos do sistema. Além disso, o Superior Scheduler combina tarefas com recursos, fornecendo a cada tarefa agendada uma visualização de recursos global e aumentando a precisão do agendamento. Comparado com o agendador de código aberto, o Superior Scheduler se destaca em taxa de transferência do sistema, uso de recursos e afinidade de dados.

Figura 6-134 Comparação do Superior Scheduler com os agendadores de código aberto



Além da taxa de transferência e da utilização aprimoradas do sistema, o Superior Scheduler oferece os seguintes principais recursos de agendamento:

- Vários pools de recursos

  Vários pools de recursos ajudam a dividir logicamente os recursos do cluster e compartilhá-los entre vários locatários ou filas. A divisão de pools de recursos suporta recursos heterogêneos. Os pools de recursos podem ser divididos exatamente de acordo com os requisitos do isolamento de recursos da aplicação. Você pode configurar políticas adicionais para filas diferentes em um pool.

- Agendamento multi-locatário (**reserve**, **min**, **share** e **max**) em cada pool de recursos

  O Superior Scheduler fornece uma política de agendamento multi-locatário hierárquica flexível. Políticas diferentes podem ser configuradas para diferentes locatários ou filas que podem acessar diferentes pools de recursos. A figura a seguir lista as políticas suportadas:

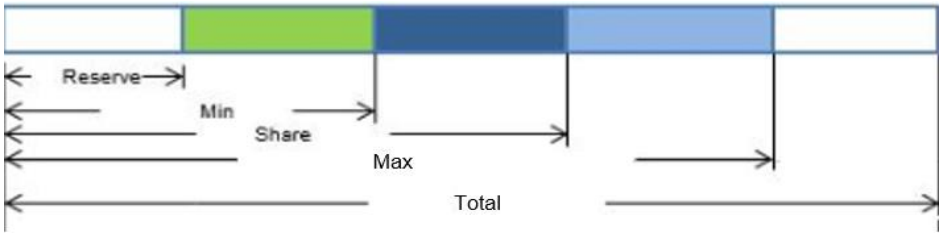
Tabela 6-29 Descrição da política

Nome	Descrição
reserve	Esta política é usada para reservar recursos para um locatário. Mesmo que o locatário não tenha trabalhos disponíveis, outro locatário não pode usar o recurso reservado. O valor pode ser uma porcentagem ou um valor absoluto. Se a porcentagem e o valor absoluto estiverem configurados, a porcentagem será calculada automaticamente em um valor absoluto e o valor maior será usado. O valor de **reserve** padrão é **0**. Em comparação com o método de especificar um pool de recursos dedicado e hosts, a política **reserve** fornece uma função de reserva flutuante flexível. Além disso, como nenhum host específico é especificado, a afinidade de dados para o cálculo é melhorada e o impacto dos hosts defeituosos é evitado.

Nome	Descrição
min	Esta política permite a preempção de recursos mínimos. Outros locatários podem usar esses recursos, mas o locatário atual tem a prioridade de usá-los. O valor pode ser uma porcentagem ou um valor absoluto. Se a porcentagem e o valor absoluto estiverem configurados, a porcentagem será calculada automaticamente em um valor absoluto e o valor maior será usado. O valor padrão é **0**.
Compartilhar	Essa política é usada para recursos compartilhados que não podem ser preemptivos. Para usar esses recursos, o locatário atual precisa esperar que outros locatários concluam trabalhos e liberem recursos. O valor pode ser uma porcentagem ou um valor absoluto.
max	Esta política é usada para o máximo de recursos que podem ser utilizados. O locatário não pode obter mais recursos do que o valor máximo permitido. O valor pode ser uma porcentagem ou um valor absoluto. Se a porcentagem e o valor absoluto estiverem configurados, a porcentagem será calculada automaticamente em um valor absoluto e o valor maior será usado. Por valor padrão, não há restrição de recursos.

Figura 6-135 mostra a política de alocação de recursos do locatário.

Figura 6-135 Políticas de agendamento de recursos



NOTA

Na figura acima, **Total** indica o número total de recursos, não a política de agendamento.

Comparado com os agendadores de código aberto, o Superior Scheduler suporta tanto a porcentagem quanto o valor absoluto de locatários para alocação de recursos, abordando com flexibilidade os requisitos de agendamento de recursos dos locatários ao nível empresarial. Por exemplo, os recursos podem ser alocados de acordo com o valor absoluto dos locatários de nível 1, evitando o impacto causado por mudanças na escala do cluster. No entanto, os recursos podem ser alocados de acordo com a porcentagem de alocação dos sublocatários, melhorando o uso de recursos no locatário de nível 1.

- Agendamento de recursos heterogêneo e multidimensional

  O Superior Scheduler suporta as seguintes funções, exceto agendamento de CPU e memória:

  - Os rótulos de nó podem ser usados para identificar atributos multidimensionais de nós, como **GPU_ENABLED** e **SSD_ENABLED**, e podem ser agendados com base nesses rótulos.

  - Os pools de recursos podem ser usados para agrupar recursos do mesmo tipo e alocá-los a locatários ou filas específicas.

- Agendamento justo de vários usuários em um locatário

  Em um locatário folha, vários usuários podem usar a mesma fila para enviar jobs. Em comparação com os agendadores de código aberto, o Superior Scheduler suporta a configuração de políticas flexíveis de compartilhamento de recursos entre diferentes usuários em um mesmo locatário. Por exemplo, os usuários VIP podem ser configurados com maior peso de acesso a recursos.

- Agendamento de reconhecimento de localidade de dados

  O Superior Scheduler adota a política de agendamento de job para nó. Ou seja, o Superior Scheduler tenta agendar jobs especificados entre os nós disponíveis para que o nó selecionado seja adequado para os jobs especificados. Ao fazer isso, o agendador terá uma visão geral do cluster e dos dados. A localização é garantida se houver uma oportunidade de colocar as tarefas mais próximas dos dados. O agendamento de código aberto usa a política de agendamento de nó para job para corresponder os jobs apropriados a um determinado nó.

- Reserva dinâmica de recursos durante o agendamento de contêiner

  Em um ambiente de computação heterogêneo e diversificado, alguns contêineres precisam de mais recursos ou vários recursos. Por exemplo, o job do Spark pode exigir grande memória. Quando tais contêineres competem com contêineres que exigem menos recursos, os contêineres que exigem mais recursos podem não obter recursos suficientes dentro de um período razoável. Agendadores de código aberto alocam recursos para jobs, o que pode causar reservas de recursos não razoáveis para esses jobs. Esse mecanismo leva ao desperdício de recursos gerais do sistema. O Superior Scheduler difere dos agendadores de código aberto nos seguintes aspectos:

  - Correspondência baseada em requisitos: o Superior Scheduler agenda jobs para os nós e seleciona os nós apropriados para reservar recursos para melhorar o tempo de inicialização dos contêineres e evitar desperdícios.
  - Reequilíbrio do locatário: quando a lógica de reserva está ativada, os agendadores de código aberto não estão em conformidade com a política de compartilhamento configurada. O Superior Scheduler usa métodos diferentes. Em cada período de programação, o Superior Scheduler percorre todos os locatários e tenta equilibrar os recursos com base na política de vários locatários. Além disso, o Superior Scheduler tenta atender a todas as políticas (**reserve**, **min** e **share**) para liberar recursos reservados e direcionar recursos disponíveis para outros contêineres que devem obter recursos sob diferentes locatários.

- Controle dinâmico de status da fila (**Open**/**Closed**/**Active**/**Inactive**)

  Vários status de fila são suportados, ajudando os administradores de cluster do MRS a gerenciar e manter vários locatários.

  - Status aberto (**Open/Closed**): se o status for **Open** por padrão, as aplicações enviadas à fila serão aceitos. Se o status for **Closed**, nenhuma aplicação será aceita.
  - Status ativo (**Active/Inactive**): se o status for **Active** por padrão, os recursos podem ser agendados e alocados para aplicações no locatário. Os recursos não serão agendados para filas no status **Inactive**.

- Motivo pendente da aplicação

  Se a aplicação não for iniciado, forneça os motivos de pendência do job.

Tabela 6-30 descreve o resultado da comparação dos agendadores de código aberto do Superior Scheduler e do Yarn.

Tabela 6-30 Análise comparativa

Agendamento	Agendador de código aberto do Yarn	Superior Scheduler
Agendamento multi-locatário	Em clusters homogêneos, o Capacity Scheduler ou o Fair Scheduler podem ser selecionados e o cluster não suporta o Fair Scheduler. O Capacity Scheduler suporta o agendamento por porcentagem e o Fair Scheduler suporta o agendamento por valor absoluto.	● Suporta clusters heterogêneos e vários pools de recursos. ● Suporta **reservation** para garantir acesso direto aos recursos.
Agendamento de reconhecimento de localidade de dados	A política de agendamento de nó para job reduz a taxa de sucesso da localização de dados e afeta potencialmente o desempenho da execução de aplicações.	A **job-to-node scheduling policy** pode reconhecer a localização de dados com mais precisão e a taxa de acerto de job do agendamento de localização de dados é maior.
Agendamento balanceado com base na carga de hosts	Incompatível	**O agendamento balanceado pode ser alcançado quando o Superior Scheduler considera a carga do host e a alocação de recursos durante o agendamento.**
Agendamento justo de vários usuários em um locatário	Incompatível	Suporta palavras-chave **default** e **others**.
Razão de espera do job	Incompatível	Razões de espera de job ilustram por que um job precisa esperar.

Em conclusão, o Superior Scheduler é um agendador de alto desempenho com várias políticas de agendamento e é melhor do que o Capacity Scheduler em termos de funcionalidade, desempenho, uso de recursos e escalabilidade.

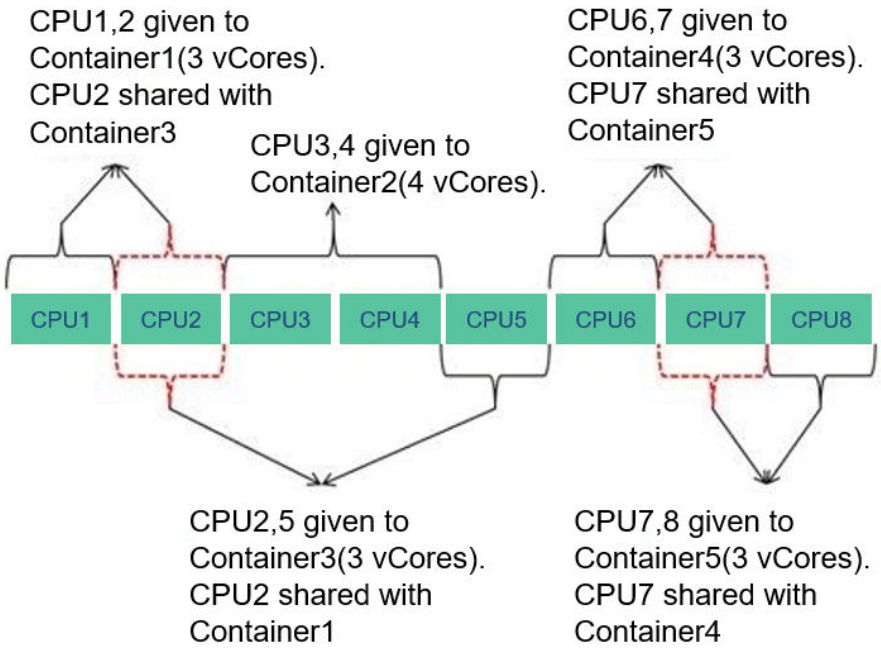
Isolamento rígido da CPU

O Yarn não pode controlar estritamente os recursos da CPU usados por cada contêiner. Quando o subsistema da CPU é usado, um contêiner pode ocupar recursos excessivos. Portanto, CPUset é usado para controlar a alocação de recursos.

Para resolver esse problema, os recursos da CPU são alocados para cada contêiner com base na proporção de núcleos virtuais (vCores) para núcleos físicos. Se um contêiner requer um núcleo físico inteiro, o contêiner o possui. Se um contêiner precisar apenas de alguns núcleos

físicos, vários contêineres podem compartilhar o mesmo núcleo físico. A figura a seguir mostra um exemplo da cota de CPU. A proporção dada de vCores para núcleos físicos é de 2:1.

Figura 6-136 Cota da CPU

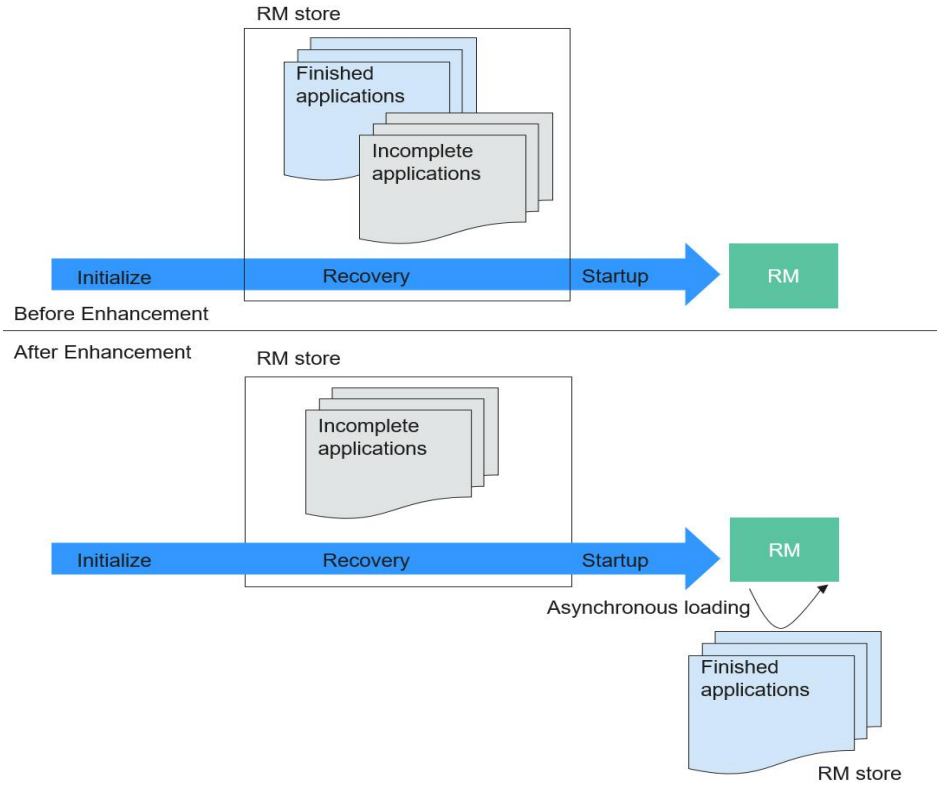


Característica de código aberto aprimorada: otimizando o desempenho da reinicialização

Geralmente, o ResourceManager recuperado pode obter aplicações em execução e concluídas. No entanto, um grande número de aplicações concluídas pode causar problemas como inicialização lenta e tempo de ResourceManagers de comutação/reinicialização de HA.

Para acelerar a inicialização, obtenha a lista de aplicações inacabadas antes de iniciar o ResourceManagers. Neste caso, a aplicação concluída continua a ser recuperada no thread assíncrono em segundo plano. A figura a seguir mostra como a recuperação do ResourceManager começa.

Figura 6-137 Iniciar a recuperação do ResourceManager



6.32 ZooKeeper

6.32.1 Princípios básicos do ZooKeeper

Visão geral

ZooKeeper é um serviço de coordenação distribuído e altamente disponível. O ZooKeeper é usado para fornecer as seguintes funções:

- Impede o sistema de SPOFs e fornece serviços confiáveis para aplicações.
- Fornece serviços de coordenação distribuídos e gerencia informações de configuração.

Arquitetura

Os nós em um cluster do ZooKeeper têm três funções: Líder, Seguidor e Observador, conforme mostrado em **Figura 6-138**. Geralmente, um número ímpar de serviços ZooKeeper (2N+1) precisa ser configurado no cluster, e pelo menos (N+1) maioria de votos é necessária para executar a operação de gravação.

Figura 6-138 Arquitetura

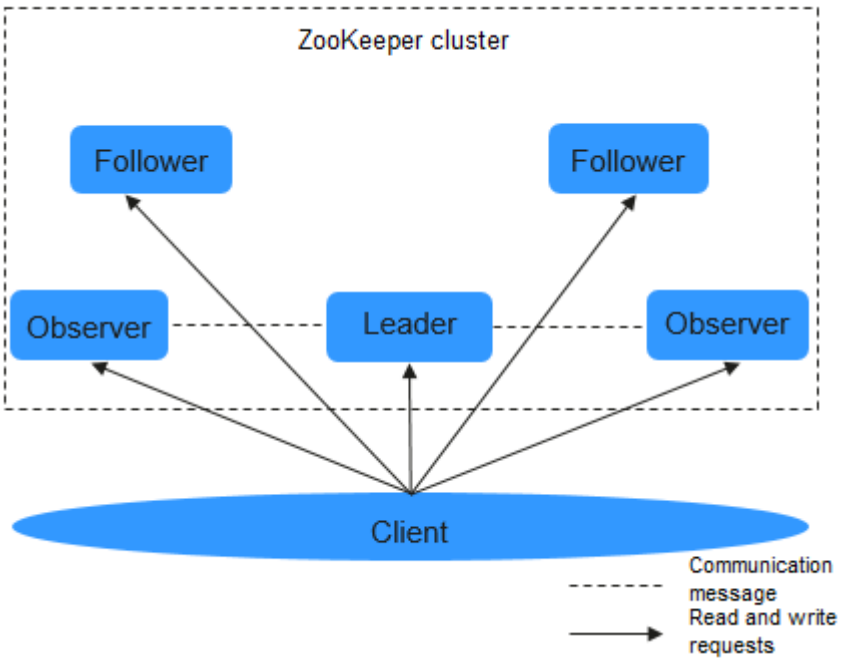


Tabela 6-31 descreve as funções de cada módulo mostrado em **Figura 6-138**.

Tabela 6-31 Descrição da arquitetura

Nome	Descrição
Líder	Apenas um nó serve como o Líder em um cluster de ZooKeeper. O Líder, eleito pelos Seguidores usando o protocolo de Transmissão Atômica do ZooKeeper (ZAB), recebe e coordena todas as solicitações de escrita e sincroniza as informações escritas para Seguidores e Observadores.
Seguidor	O Seguidor tem duas funções: ● Previne SPOFs. Um novo Líder é eleito dos Seguidores quando o Líder está com defeito. ● Processa solicitações de leitura e interage com o Líder para processar solicitações de gravação.
Observador	O observador não participa na votação para a eleição e escreve solicitações. Ele processa apenas solicitações de leitura e encaminha solicitações de gravação para o Líder, aumentando a eficiência do processamento do sistema.
Cliente	Lê e grava dados de ou para o cluster do ZooKeeper. Por exemplo, o HBase pode servir como um cliente de ZooKeeper e usar a função de arbitragem do cluster do ZooKeeper para controlar o status ativo/em espera do HMaster.

Se os serviços de segurança estiverem ativados no cluster, a autenticação será necessária durante a conexão com a ZooKeeper. Os modos de autenticação são os seguintes:

- Modo de Keytab: você precisa obter um usuário homem-máquina do administrador de cluster do MRS para logon e autenticação do console do MRS e obter o arquivo Keytab do usuário.
- Modo de tíquete: obtenha um usuário humano-máquina do administrador de cluster do MRS para logon seguro subsequente, habilite as funções renováveis e encaminhadas do serviço Kerberos, defina o período de atualização do tíquete e reinicie o Kerberos e os componentes relacionados.

  NOTA

  - Por padrão, o período de validade da senha do usuário é de 90 dias. Portanto, o período de validade do arquivo Keytab obtido é de 90 dias.
  - Os parâmetros para ativar as funções renováveis e encaminhadas e definir o intervalo de atualização do tíquete estão na guia **System** da página de configuração do serviço Kerberos. O intervalo de atualização do tíquete pode ser definido como **kdc_renew_lifetime** ou **kdc_max_renewable_life** com base na situação real.

Princípios

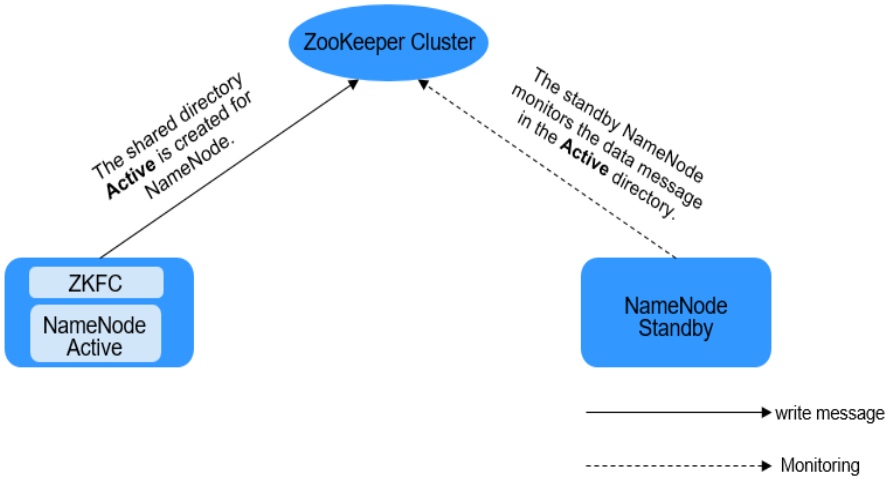
- **Solicitação de gravação**
  a. Depois que o Seguidor ou Observador recebe uma solicitação por escrito, o Seguidor ou Observador envia a solicitação ao Líder.
  b. O Líder coordena os Seguidores para determinar se aceita o pedido de escrita por votação.
  c. Se mais da metade dos eleitores retornarem uma mensagem de sucesso de gravação, o líder enviará a solicitação de gravação e retornará uma mensagem de sucesso. Caso contrário, uma mensagem de falha é retornada.
  d. O Seguidor ou Observador retorna os resultados de processamento.
- **Solicitação somente leitura**
  O cliente lê diretamente os dados do Líder, Seguidor ou Observador.

6.32.2 Relação entre ZooKeeper e outros componentes

Relação entre ZooKeeper e HDFS

Figura 6-139 mostra a relação entre o ZooKeeper e o HDFS.

Figura 6-139 Relação entre ZooKeeper e HDFS



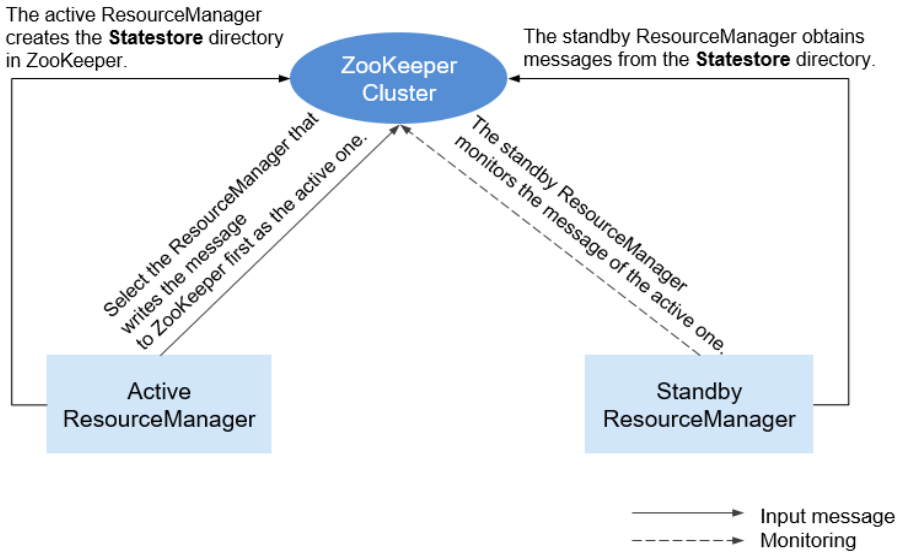
Como cliente de um cluster de ZooKeeper, o ZKFailoverController (ZKFC) monitora o status do NameNode. O ZKFC é implementado apenas no nó em que o NameNode reside e nos NameNodes HDFS ativos e em espera.

1. O ZKFC se conecta ao ZooKeeper e salva informações como nomes de host em ZooKeeper no diretório de znode **/hadoop-ha**. O NameNode que cria o diretório primeiro é considerado como o nó ativo e o outro é o nó em espera. NameNodes lê as informações do NameNode periodicamente através do ZooKeeper.
2. Quando o processo do nó ativo termina anormalmente, o NameNode em espera detecta alterações no diretório **/hadoop-ha** por meio do ZooKeeper e, em seguida, assume o serviço do NameNode ativo.

Relação entre ZooKeeper e YARN

Figura 6-140 mostra a relação entre ZooKeeper e YARN.

Figura 6-140 Relação entre ZooKeeper e YARN

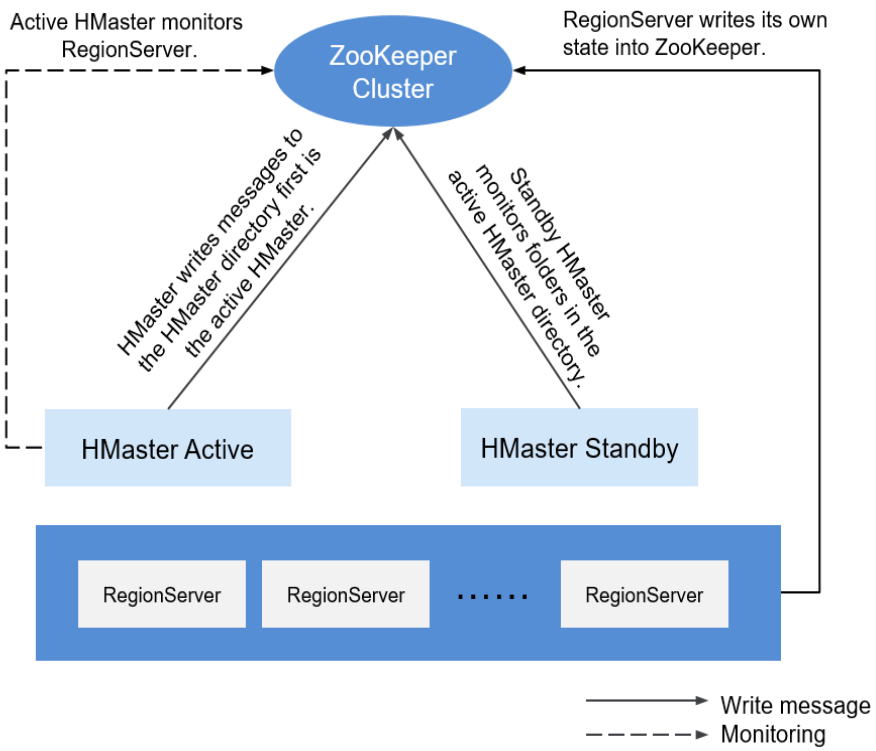


1. Quando o sistema é iniciado, o ResourceManager tenta gravar informações de estado no ZooKeeper. O ResourceManager que primeiro grava informações de estado no ZooKeeper é selecionado como o ResourceManager ativo e outros são ResourceManagers em espera. Os ResourceManagers em espera monitoram periodicamente as informações eleitorais ativas do ResourceManager no ZooKeeper.
2. O ResourceManager ativo cria o diretório **Statestore** no ZooKeeper para armazenar informações da aplicação. Se o ResourceManager ativo estiver com defeito, o ResourceManager em espera obterá informações da aplicação do diretório **Statestore** e restaurará os dados.

Relação entre ZooKeeper e HBase

Figura 6-141 mostra a relação entre o ZooKeeper e o HBase.

Figura 6-141 Relação entre ZooKeeper e HBase

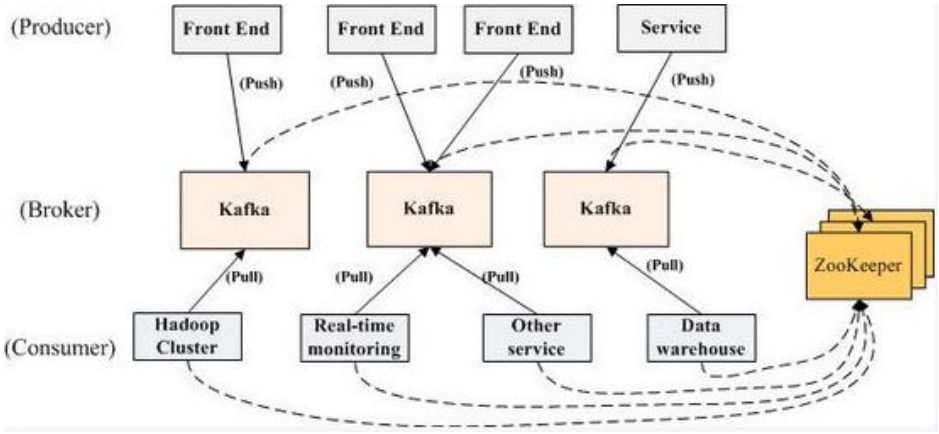


1. O HRegionServer registra-se em ZooKeeper no nó Efêmero. O ZooKeeper armazena as informações do HBase, incluindo os metadados do HBase e os endereços de HMaster.
2. HMaster detecta o status de saúde de cada HRegionServer usando o ZooKeeper e os monitora.
3. O HBase oferece suporte a vários nós HMaster (como NameNodes do HDFS). Quando o HMaster ativo está com defeito, o HMaster em espera obtém as informações de estado sobre todo o cluster usando ZooKeeper. Ou seja, o uso de ZooKeeper pode evitar SPOFs de HBase.

Relação entre ZooKeeper e Kafka

Figura 6-142 mostra a relação entre ZooKeeper e Kafka.

Figura 6-142 Relação entre ZooKeeper e Kafka



1. O agente usa o ZooKeeper para registrar informações do agente e eleger um líder da partição.
2. O consumidor usa o ZooKeeper para registrar informações do consumidor, incluindo a lista de partição do consumidor. Além disso, o ZooKeeper é usado para descobrir a lista de agentes, estabelecer uma conexão de soquete com o líder da partição e obter mensagens.

6.32.3 Recursos de código aberto aprimorados do ZooKeeper

Log aprimorado

No modo de segurança, um nó efêmero é excluído desde que a sessão que criou o nó expire. A exclusão de nó efêmero é registrada nos logs de auditoria para que o status de nó efêmero possa ser obtido.

Os nomes de usuário devem ser adicionados aos logs de auditoria de todas as operações executadas em clientes do ZooKeeper.

No cliente do ZooKeeper, crie um znode, do qual o principal do Kerberos é **zkcli/hadoop.**<*System domain name*>**@**<*System domain name*>.

Por exemplo, abra o arquivo **<ZOO_LOG_DIR>/zookeeper_audit.log**. O conteúdo do arquivo é o seguinte:

```
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
```

O conteúdo mostra que os logs do usuário cliente do ZooKeeper **zkcli/hadoop.hadoop.com@HADOOP.COM** são adicionados ao log de auditoria.

Detalhes do usuário no ZooKeeper

No ZooKeeper diferentes esquemas de autenticação usam credenciais diferentes como usuários. Com base no requisito do provedor de autenticação, qualquer parâmetro pode ser considerado como usuário.

Exemplo:

- **SAMLAuthenticationProvider** usa o cliente principal como um usuário.
- **X509AuthenticationProvider** usa o certificado de cliente do usuário como um usuário.
- **IAuthenticationProvider** usa o endereço IP do cliente como um usuário.
- Um nome de usuário pode ser obtido do provedor de autenticação personalizado implementando o método **org.apache.zookeeper.server.auth.ExtAuthenticationProvider.getUserName(String)**. Se o método não for implementado, a obtenção do nome de usuário da instância do provedor de autenticação será ignorada.

Recurso de código aberto aprimorado: comunicação SSL do ZooKeeper (Conexão Netty)

O design do ZooKeeper contém o pacote Nio e não suporta SSL posterior à versão 3.5. Para resolver este problema, o Netty é adicionado ao ZooKeeper. Portanto, se você precisar usar SSL, ative Netty e defina os seguintes parâmetros no servidor e no cliente:

O servidor de código aberto suporta apenas senhas de texto simples, o que pode causar problemas de segurança. Portanto, essas senhas de texto não são mais usadas no servidor.

- Cliente
  a. Defina **-Dzookeeper.client.secure** no arquivo **zkCli.sh/zkEnv.sh** como **true** para usar a comunicação segura no cliente. Em seguida, o cliente pode se conectar ao SecureClientPort no servidor.
  b. Defina os seguintes parâmetros no arquivo **zkCli.sh/zkEnv.sh** para configurar o ambiente cliente:

Parâmetro	Descrição
-Dzookeeper.clientCnxnSocket	Usado para comunicação Netty entre clientes. Valor padrão: **org.apache.zookeeper.ClientCnxnSocketNetty**

Parâmetro	Descrição
-Dzookeeper.ssl.keyStore.location	Indica o caminho para armazenar o arquivo keystore.
-Dzookeeper.ssl.keyStore.password	Criptografa uma senha.
-Dzookeeper.ssl.trustStore.location	Indica o caminho para armazenar o arquivo truststore.
-Dzookeeper.ssl.trustStore.password	Criptografa uma senha.
-Dzookeeper.config.crypt.class	Descriptografa uma senha criptografada.
-Dzookeeper.ssl.password.encrypted	Valor padrão: **false** Se as senhas de armazenamento de chaves e de armazenamento de confiança estiverem criptografadas, defina esse parâmetro como **true**.
-Dzookeeper.ssl.enabled.protocols	Define os protocolos SSL a serem ativados para o contexto SSL.
-Dzookeeper.ssl.exclude.cipher.ext	Define a lista de senhas separadas por uma vírgula que devem ser excluídas do contexto SSL.

NOTA

Os parâmetros anteriores devem ser definidos no arquivo **zkCli.sh/zk.Env.sh**.

- Servidor
  a. Defina **secureClientPort** como **3381** no arquivo **zoo.cfg**.
  b. Defina **zookeeper.serverCnxnFactory** como **org.apache.zookeeper.server.NettyServerCnxnFactory** no arquivo **zoo.cfg** no servidor.
  c. Defina os seguintes parâmetros no arquivo **zoo.cfg** (no caminho **zookeeper/conf/zoo.cfg**) para configurar o ambiente de servidor:

Parâmetro	Descrição
ssl.keyStore.location	Caminho para armazenar o arquivo **keystore.jks**
ssl.keyStore.password	Criptografa uma senha.
ssl.trustStore.location	Indica o caminho para armazenar o arquivo truststore.
ssl.trustStore.password	Criptografa uma senha.

Parâmetro	Descrição
config.crypt.class	Descriptografa uma senha criptografada.
ssl.keyStore.password.encrypted	Valor padrão: **false** Se esse parâmetro for definido como **true**, a senha criptografada poderá ser usada.
ssl.trustStore.password.encrypted	Valor padrão: **false** Se esse parâmetro for definido como **true**, a senha criptografada poderá ser usada.
ssl.enabled.protocols	Define os protocolos SSL a serem ativados para o contexto SSL.
ssl.exclude.cipher.ext	Define a lista de senhas separadas por uma vírgula que devem ser excluídas do contexto SSL.

d. Inicie o ZKserver e conecte o cliente de segurança à porta de segurança.

- Credencial

  A credencial usada entre o cliente e o servidor no ZooKeeper é **X509AuthenticationProvider**. Essa credencial é inicializada usando os certificados de servidor especificados e confiáveis pelos seguintes parâmetros:

  - zookeeper.ssl.keyStore.location
  - zookeeper.ssl.keyStore.password
  - zookeeper.ssl.trustStore.location
  - zookeeper.ssl.trustStore.password

  NOTA

  Se você não quiser usar o mecanismo padrão do ZooKeeper ele pode ser configurado com diferentes mecanismos de confiança conforme necessário.

7 Funções

7.1 Multi-locatário

Introdução de recursos

Os clusters de dados das empresas modernas estão se desenvolvendo em direção à centralização e à cloudificação. Os clusters de Big Data de classe empresarial devem atender aos seguintes requisitos:

- Transportar dados de diferentes tipos e formatos e executar jobs e aplicações de diferentes tipos (análise, consulta e processamento de fluxo).
- Isolar os dados de um usuário daqueles de outro usuário que tenha requisitos exigentes de segurança de dados, como um banco ou instituto governamental.

Os requisitos anteriores trazem os seguintes desafios para o cluster de Big Data:

- Alocação adequada e programação de recursos para garantir o funcionamento estável de aplicações e jobs
- Controle de acesso rigoroso para garantir a segurança de dados e serviços

O multi-locatário isola os recursos de um cluster de Big Data em conjuntos de recursos. Os usuários podem alugar conjuntos de recursos desejados para executar aplicações e jobs e armazenar dados. Em um cluster de Big Data, vários conjuntos de recursos podem ser implementados para atender a diversos requisitos de vários usuários.

O cluster de Big Data do MRS fornece uma solução multi-locatário de Big Data de classe empresarial completa. Multi-locatário é uma coleção de vários recursos (cada conjunto de recursos é um locatário) em um cluster de Big Data do MRS. Ele pode alocar e agendar recursos, incluindo recursos de computação e armazenamento.

Vantagens

- Configuração e isolamento adequados dos recursos

  Os recursos de um locatário são isolados dos de outro locatário. O uso de recursos de um locatário não afeta outros locatários. Esse mecanismo garante que cada locatário possa configurar recursos com base nos requisitos de serviço, melhorando a utilização de recursos.

- Medição e estatísticas do consumo de recursos

  Os locatários são candidatos a recursos do sistema e consumidores. Os recursos do sistema são planejados e alocados com base nos locatários. O consumo de recursos pelos inquilinos pode ser medido e registrado.

- Garantida segurança de dados e segurança de acesso

  Em cenários de vários locatários, os dados de cada locatário são armazenados separadamente para garantir a segurança dos dados. O acesso aos recursos dos locatários é controlado para garantir a segurança do acesso.

Agendadores aprimorados

Os agendadores são divididos no Capacity scheduler de código aberto e no Superior scheduler de propriedade da Huawei .

Para atender aos requisitos da empresa e enfrentar os desafios enfrentados pela comunidade Yarn em programação, Huawei desenvolve o Superior scheduler. Além de herdar as vantagens do Capacity scheduler e do Fair scheduler, esse agendador é aprimorado nos seguintes aspectos:

- Política de compartilhamento de recursos aprimorada

  O Superior scheduler suporta hierarquia de filas. Ele integra as funções de agendadores de código aberto e compartilha recursos com base em políticas configuráveis. Em termos de instâncias, os administradores de cluster do MRS podem usar o Superior scheduler para configurar um valor absoluto ou uma política de porcentagem para recursos de fila. A política de compartilhamento de recursos do Superior scheduler aprimora a política de agendamento de rótulos do Yarn como um recurso de pool de recursos. Os nós no cluster do Yarn podem ser agrupados com base na capacidade ou no tipo de serviço para garantir que as filas possam utilizar os recursos com mais eficiência.

- Política de reserva de recursos baseada em locatários

  Os recursos exigidos pelos locatários devem ser assegurados para a execução de tarefas críticas. O Superior scheduler constrói um mecanismo para apoiar a política de reserva de recursos. Ao fazer isso, os recursos reservados podem ser alocados para as tarefas executadas pelas filas de locatários em tempo hábil para garantir a execução adequada da tarefa.

- Partilha justa entre locatários e usuários do pool de recursos

  O Agendador Superior permite que recursos compartilhados sejam configurados para usuários em uma fila. Cada locatário pode ter usuários com pesos diferentes. Usuários altamente ponderados podem precisar de mais recursos compartilhados.

- Desempenho de agendamento garantido em um grande cluster

  O Superior scheduler recebe pulsações de cada NodeManager e salva informações de recursos na memória, o que permite controlar o uso de recursos do cluster globalmente. O Superior scheduler usa o modelo de agendamento push, o que torna o agendamento mais preciso e eficiente e melhora notavelmente a utilização de recursos do cluster. Além disso, o Superior scheduler oferece excelente desempenho quando o intervalo entre batimentos cardíacos do NodeManager é longo e evita tempestades de batimentos cardíacos em grandes clusters.

- Política de prioridade

  Se o requisito de recurso mínimo de um serviço não puder ser atendido após o serviço obter todos os recursos disponíveis, ocorrerá uma preempção. A função de preempção está desativada por padrão.

7.2 Fortalecimento de segurança

O MRS é uma plataforma para gerenciamento e análise de dados massivos e possui alta segurança. O MRS protege os dados do usuário e o serviço em execução a partir dos seguintes aspectos:

- Isolamento de rede

  Todo o sistema é implantado em uma VPC na nuvem pública para fornecer um ambiente de rede isolado e garantir a segurança do serviço e do gerenciamento do cluster. Combinando as funções de divisão de sub-rede, controle de rota e grupo de segurança da VPC, o MRS fornece um ambiente de rede isolado seguro e confiável.

- Isolamento de recursos

  O MRS suporta a implantação de recursos e o isolamento de recursos físicos em zonas dedicadas. Você pode combinar recursos de computação e armazenamento de forma flexível, como recursos de computação dedicados + recursos de armazenamento compartilhados, recursos de computação compartilhados + recursos de armazenamento dedicados e recursos de computação dedicados + recursos de armazenamento dedicados.

- Segurança de host

  O MRS pode ser integrado a serviços de segurança de nuvem pública, incluindo Vulnerability Scan Service (VSS), Host Security Service (HSS), Web Application Firewall (WAF), Cloud Bastion Host (CBH) e Web Tamper Protection (WTP). As seguintes medidas são fornecidas pela Huawei Cloud para melhorar a segurança do sistema operacional e das portas:

  - Fortalecimento da segurança dos kernels do SO
  - Controle de permissão do SO
  - Gerenciamento de porta do SO

- Segurança de aplicações

  As seguintes medidas são usadas para garantir o funcionamento normal dos serviços de Big Data:

  - Identificação e autenticação
  - Segurança de aplicações da Web
  - Controle de acesso
  - Auditoria de segurança
  - Segurança de senha

- Segurança de dados

  As seguintes medidas são fornecidas para garantir a confidencialidade, integridade e disponibilidade de grandes quantidades de dados do usuário:

  - Recuperação de desastres: o MRS suporta backup de dados para OBS e alta confiabilidade entre regiões.
  - Backup: o MRS suporta backup de metadados DBService, NameNode e LDAP e backup de dados de serviço HDFS e HBase.

- Integridade dos dados

  Os dados são verificados para garantir sua integridade durante o armazenamento e a transmissão.

  - O CRC32C é usado por padrão para verificar a exatidão dos dados do usuário armazenados no HDFS.
  - DataNodes do HDFS armazenam os dados verificados. Se os dados transmitidos de um cliente são anormais (incompletos), DataNodes informa a anormalidade ao cliente e o cliente reescreve os dados.
  - O cliente verifica a integridade dos dados ao ler dados de um DataNode. Se os dados estiverem incompletos, o cliente lerá os dados de outro DataNode.
- Confidencialidade de dados

  Baseado no Apache Hadoop, o sistema de arquivos distribuído do MRS suporta armazenamento criptografado de arquivos para evitar que dados confidenciais sejam armazenados em texto simples, melhorando a segurança dos dados. As aplicações precisam apenas criptografar dados confidenciais especificados. Os serviços não são afetados durante o processo de criptografia. Com base na criptografia de dados do sistema de arquivos, o Hive fornece criptografia em nível de tabela e o HBase fornece criptografia em nível de família de colunas. Dados sensíveis podem ser criptografados e armazenados depois que você especificar um algoritmo de criptografia durante a criação da tabela.

  O armazenamento criptografado e o controle de acesso aos dados são usados para garantir a segurança dos dados do usuário.
  - O HBase armazena dados de serviço no HDFS após a compactação. Os usuários podem configurar o algoritmo de criptografia AES e SMS4 para criptografar dados.
  - Todos os componentes permitem que permissões de acesso sejam definidas para diretórios de dados locais. Usuários não autorizados não têm permissão para acessar dados.
  - Todas as informações do usuário do cluster são armazenadas em texto cifrado.
- Autenticação de segurança
  - Usa um sistema unificado de autenticação baseado em usuário e função, bem como um modelo de controle de acesso baseado em conta e função (RBAC) para controlar centralmente as permissões de usuário e gerenciar em lote autorização de usuário.
  - Emprega o Potocolo Lightweight Directory Access (LDAP) como um sistema de gerenciamento de contas e executa a autenticação Kerberos em contas.
  - Fornece a função de logon único (SSO) que gerencia e autentica centralmente os usuários do sistema e do componente do MRS.
  - Audita os usuários que fizeram logon no Manager.

7.3 Acesso fácil a IUs da Web de componentes

Os componentes de Big Data têm suas próprias IUs da Web para gerenciar seus próprios sistemas. No entanto, você não pode acessar facilmente as IUs da Web devido ao isolamento de rede. Por exemplo, para acessar a IU da Web do HDFS, você precisa criar um ECS para efetuar logon remotamente na IU da Web. Isso torna o acesso à interface do usuário complexo e hostil.

O MRS fornece um canal seguro baseado em EIP para você acessar facilmente as IUs da Web dos componentes. Isso é mais conveniente do que vincular um EIP por conta própria, e você pode acessar as IUs da Web com alguns cliques, evitando as etapas de logon em uma VPC, adicionando regras de grupo de segurança e obtendo um endereço IP público. Para os componentes Hadoop, Spark, HBase e Hue em clusters de análise e o componente Storm em

clusters de streaming, você pode acessar rapidamente suas IUs da Web a partir das entradas no Manager.

7.4 Aprimoramento da confiabilidade

Baseado no software de código aberto Apache Hadoop, o MRS otimiza e melhora a confiabilidade e o desempenho dos principais componentes do serviço.

Confiabilidade do sistema

- HA para todos os nós de gerenciamento

  Na versão de código aberto do Hadoop, os dados e nós de computação são gerenciados em um sistema distribuído, no qual um único ponto de falha (SPOF) não afeta a operação de todo o sistema. No entanto, um SPOF pode ocorrer em nós de gerenciamento em execução no modo centralizado, o que se torna o ponto fraco da confiabilidade geral do sistema.

  O MRS fornece mecanismos semelhantes de nó duplo para todos os nós de gerenciamento dos componentes de serviço, como Manager, HDFS NameNodes, HiveServers, HBase HMasters, Yarn ResourceManagers, KerberosServers e LdapServers. Todos eles são implementados no modo ativo/em espera ou configurados com compartilhamento de carga, impedindo efetivamente que os SPOFs afetem a confiabilidade do sistema.

- Garantia de confiabilidade em caso de excepções

  Por análise de confiabilidade, as seguintes medidas para lidar com exceções de software e hardware são fornecidas para melhorar a confiabilidade do sistema:

  - Depois que a fonte de alimentação é restaurada, os serviços funcionam adequadamente, independentemente de uma falha de energia de um único nó ou de todo o cluster, garantindo a confiabilidade dos dados em caso de falhas de energia inesperadas. Os dados-chave não serão perdidos a menos que o disco rígido esteja danificado.
  - As verificações de status de integridade e o tratamento de falhas do disco rígido não afetam os serviços.
  - As falhas do sistema de arquivos podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.
  - As falhas de processo e nó podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.
  - As falhas de rede podem ser tratadas automaticamente e os serviços afetados podem ser restaurados automaticamente.

- Backup e restauração de dados

  O MRS fornece funções de backup completo, backup incremental e restauração com base nos requisitos de serviço, evitando o impacto da perda de dados e danos nos serviços e garantindo a restauração rápida do sistema em caso de exceções.

  - Backup automático

    O MRS fornece backup automático para dados no Manager. Com base na política de backup personalizada, os dados em clusters, incluindo no LdapServer e DBService, podem ser automaticamente copiados.

  - Backup manual

Você também pode fazer backup manual dos dados do sistema de gerenciamento de cluster antes da expansão de capacidade e instalação de patch para recuperar as funções do sistema de gerenciamento de cluster em caso de falhas.

Para melhorar a confiabilidade do sistema, o backup dos dados no Manager e no HBase é feito manualmente em um servidor de terceiros.

Confiabilidade do nó

- Monitoramento do status de integridade do sistema operacional

  O MRS coleta periodicamente dados de uso de recursos de hardware do SO, incluindo o uso de CPUs, memória, discos rígidos e recursos de rede.

- Monitoramento do status de integridade do processo

  O MRS verifica o status das instâncias de serviço e os indicadores de integridade dos processos da instância de serviço, permitindo que você conheça o status de integridade dos processos em tempo hábil.

- Solução automática de problemas de disco

  O MRS é aprimorado com base na versão de código aberto. Ele pode monitorar o status do hardware e dos sistemas de arquivos em todos os nós. Se ocorrer uma exceção, as partições correspondentes serão removidas do pool de armazenamento. Se um disco estiver com defeito e for substituído, um novo disco rígido será adicionado para executar serviços. Neste caso, as operações de manutenção são simplificadas. A substituição de discos defeituosos pode ser concluída on-line. Além disso, os usuários podem definir discos de backup quentes para reduzir o tempo de restauração do disco defeituoso e melhorar a confiabilidade do sistema.

- Configuração de LVM para discos de nó

  O MRS permite configurar o Gerenciamento de volume lógico (LVM) para planejar vários discos como um grupo de volumes lógicos. A configuração do LVM pode evitar o uso irregular de discos. É especialmente importante garantir o uso uniforme de discos em componentes que podem usar vários recursos de disco, como HDFS e Kafka. Além disso, o LVM suporta expansão de capacidade de disco sem reanexar, evitando a interrupção do serviço.

Confiabilidade de dados

O MRS pode usar os recursos de grupos de nós antiafinidade e grupos de posicionamento fornecidos pelo ECS e o recurso de reconhecimento de rack do Hadoop para distribuir dados de forma redundante para vários computadores host físicos, evitando a perda de dados causada por falhas de hardware físico.

7.5 Gerenciamento de job

A função de gerenciamento de job fornece uma entrada para você enviar jobs em um cluster, incluindo jobs de MapReduce e SparkSQL. O MRS trabalha com DataArts Studio da Huawei Cloud para fornecer um ambiente de desenvolvimento de colaboração de Big Data e capacidades de agendamento de Big Data totalmente gerenciadas, ajudando você a criar centros de processamento de Big Data sem esforço.

DataArts Studio permite que você desenvolva e depure scripts de MRS HiveQL/SparkSQL on-line e desenvolva jobs de MRS executando operações de arrastar e soltar para migrar e integrar dados entre MRS e mais de 20 fontes de dados heterogêneas. O poderoso

agendamento de job e o monitoramento flexível e o alarme ajudam você a gerenciar facilmente dados e jobs de O&M.

7.6 Ações de inicialização

Introdução da característica

O MRS fornece clusters de Big Data elásticos padrão na nuvem. Nove componentes de Big Data, como Hadoop e Spark, podem ser instalados e implantados. Atualmente, os clusters de Big Data em nuvem padrão não podem atender a todos os requisitos do usuário, por exemplo, nos seguintes cenários:

- As configurações comuns do sistema operacional não podem atender aos requisitos de processamento de dados, por exemplo, aumentando o número máximo de conexões do sistema.
- Ferramentas de software ou ambientes em execução precisam ser instalados, por exemplo, pacote de linguagem Gradle e dependência R.
- Os pacotes de componentes de Big Data precisam ser modificados com base nos requisitos de serviço, por exemplo, modificando o pacote de instalação do Hadoop ou do Spark.
- Outros componentes de Big Data que não são suportados pelo MRS precisam ser instalados.

Para atender aos requisitos de personalização anteriores, você pode executar operações manualmente nos nós existentes e recém-adicionados. O processo geral é complexo e propenso a erros. Além disso, as operações manuais não podem ser rastreadas e os dados não podem ser processados imediatamente após a criação de um cluster com base na sua demanda.

Portanto, o MRS oferece suporte a ações de inicialização personalizadas que permitem executar scripts em um nó especificado antes ou depois de um componente de cluster ser iniciado. Você pode executar ações de inicialização para instalar software de terceiros que não é suportado pelo MRS, modificar o ambiente de execução do cluster e executar outras personalizações. Se você optar por executar ações de inicialização ao expandir um cluster, as ações de inicialização serão executadas nos nós recém-adicionados da mesma maneira. O MRS executa o script que você especificar como usuário **root**. Você pode executar o comando **su - *xxx*** no script para alternar o usuário.

Benefícios aos clientes

Você pode usar as ações de inicialização personalizadas para configurar de forma flexível e fácil seus clusters dedicados e personalizar a instalação do software.

7.7 Enterprise Project Management

Um projeto corporativo é um modo de gerenciamento de recurso em nuvem. O Enterprise Management fornece aos usuários um gerenciamento abrangente de recursos, pessoal, permissões e finanças baseados em nuvem. Os consoles de gerenciamento comuns são orientados para o controle e a configuração de produtos individuais de nuvem. O console do Enterprise Management, em contraste, é mais focado no gerenciamento de recursos. Ele é projetado para ajudar as empresas a gerenciar recursos, pessoal, permissões e finanças baseados em nuvem, de maneira hierárquica, como gerenciamento de empresas, departamentos e projetos.

O MRS permite que os usuários que ativaram o Enterprise Project Management Service (EPS) para configurar projetos empresariais para um cluster durante a criação do cluster e usem o EPS para gerenciar recursos do MRS por grupo.

- Os usuários podem gerenciar vários recursos por grupo.
- Os usuários podem exibir informações sobre recursos e detalhes de dispêndio de projetos da empresa.
- Os usuários podem controlar as permissões de acesso no nível do projeto corporativo.
- Os usuários podem exibir informações financeiras detalhadas por projeto da empresa, incluindo ordens, resumo de dispêndio e detalhes de dispêndio.

7.8 Metadados

O MRS fornece vários métodos de armazenamento de metadados. Ao implementar o Hive e o Ranger durante a criação do cluster do MRS, selecione um dos seguintes modos de armazenamento, conforme necessário:

- **Local**: os metadados são armazenados no GaussDB local de um cluster. Quando o cluster é excluído, os metadados também são excluídos. Para reter os metadados, faça backup manual dos metadados no banco de dados com antecedência.
- **Conexão de dados externa**: você pode selecionar **RDS PostgreSQL database** ou **RDS MySQL database** que está associado à mesma VPC e sub-rede que o cluster atual. Os metadados são armazenados no banco de dados e não são excluídos quando o cluster atual é excluído. Vários clusters de MRS podem compartilhar os mesmos metadados.

NOTA

Hive no MRS 1.9.*x* ou posterior permite especificar um método de armazenamento de metadados.

O Ranger no MRS 1.9.*x* permite que os metadados sejam armazenados apenas no banco de dados MySQL associado do serviço RDS.

7.9 Gerenciamento de cluster

7.9.1 Gerenciamento do ciclo de vida do cluster

O MRS oferece suporte ao gerenciamento do ciclo de vida do cluster, incluindo a criação e o encerramento de clusters.

- Criar um cluster: depois de especificar um tipo de cluster, componentes, número de nós de cada tipo, especificações de VM, AZ, VPC e informações de autenticação, o MRS cria automaticamente um cluster que atende aos requisitos de configuração. Você pode executar scripts personalizados no cluster. Além disso, você pode criar clusters de tipos diferentes para vários cenários de aplicações, como clusters de análise do Hadoop, clusters do HBase e clusters do Kafka. A plataforma de Big Data suporta implantação de cluster heterogêneo. Ou seja, VMs de especificações diferentes podem ser combinadas em um cluster com base em tipos de CPU, capacidades de disco, tipos de disco e tamanhos de memória. Várias especificações de VM podem ser misturadas em um cluster.
- Finalizar um cluster: você pode encerrar um cluster de pagamento por uso que não seja mais necessário (incluindo dados e configurações no cluster). O MRS excluirá todos os recursos relacionados ao cluster.

- Renovação: o MRS oferece dois modos de cobrança: pagamento por uso e anual/mensal. No modo de pagamento por uso, as taxas são deduzidas a cada hora e o saldo insuficiente pode levar a pagamentos em atraso. No modo anual/mensal, os clusters precisam ser renovados antes de expirarem. Se sua assinatura para o cluster de pagamento por uso ou anual/mensal não for renovada, seus serviços continuarão em execução, mas entrarão em um período de retenção, durante o qual os clusters de MRS deixarão de ser executados, mas os dados serão retidos.
- Cancelamento de assinatura: se você tiver comprado um cluster anual/mensal e não precisar dos recursos de cluster antes que os recursos de cluster expirem, poderá cancelar a assinatura dos recursos de cluster no MRS.

Comprar um cluster

No console de gerenciamento do MRS, você pode comprar um cluster do MRS em uma base de pagamento por uso ou anual/mensal. Você pode selecionar uma região e especificações de recursos de nuvem para comprar um cluster de MRS adequado para serviços corporativos com um clique. O MRS instala e implementa automaticamente a plataforma de Big Data em nível empresarial da Huawei Cloud e otimiza os parâmetros com base no tipo e versão de cluster e nas especificações de nó selecionadas.

O MRS fornece clusters de Big Data totalmente gerenciados. Ao criar um cluster, você pode definir um modo de logon da VM (senha ou par de chaves). Você pode usar todos os recursos do cluster do MRS criado. Além disso, o MRS permite implantar um cluster de Big Data em apenas dois ECSs com 4 vCPUs e 8 GB de memória, fornecendo opções mais flexíveis para testes e desenvolvimento.

Os clusters de MRS são classificados em clusters de análise, streaming e híbridos.

- Cluster de análise: usado para análise de dados off-line e fornece componentes Hadoop.
- Cluster de streaming: é usado para tarefas de streaming e fornece componentes de processamento de fluxo.
- Cluster híbrido: utilizado não apenas para a análise de dados offline, mas também para o processamento de streaming, e fornece componentes do Hadoop e componentes de processamento de streaming.
- Personalizado: você pode combinar de forma flexível os componentes necessários (MRS 3.x e versões posteriores) com base nos requisitos de serviço.

Os nós do cluster do MRS são classificados em nós principal, central e de tarefa.

- Nó principal: nó de gerenciamento em um cluster. Processos principais de um sistema distribuído, Manager e bancos de dados são implementados em nós principais. Os nós principais não podem ser expandidos. A capacidade de processamento desses nós determina o limite superior da capacidade de gerenciamento de todo o cluster. MRS possibilita a ampliação das especificações do nó principal para fornecer suporte ao gerenciamento de um cluster maior.
- Nó central: é usado tanto para armazenamento como para computação, e pode ser expandido ou reduzido. Já que os nós centrais admitem o armazenamento de dados, impõem-se diversas restrições à redução a fim de se evitar a perda de dados, e o dimensionamento automático não pode ser realizado.
- Nó de tarefa: usado apenas para computação e pode ser expandido ou reduzido. Os nós de tarefa admitem apenas tarefas de computação. Portanto, o dimensionamento automático é possível.

Você pode comprar um cluster em dois modos: compra personalizada e compra rápida.

- Compra personalizada: na página **Custom Config**, você pode configurar de forma flexível os parâmetros de cluster com base em cenários de aplicações, como o modo de cobrança e as especificações do ECS, para melhor atender às suas necessidades de serviço.
- Compra rápida: na página **Quick Config**, você pode comprar rapidamente um cluster com base em cenários de aplicações, melhorando a eficiência da configuração do cluster. Atualmente, os clusters de análise do Hadoop, os clusters do HBase e os clusters do Kafka estão disponíveis para sua compra rápida.
  - Cluster de análise do Hadoop: usa componentes no ecossistema Hadoop de código aberto para analisar e consultar vastas quantidades de dados. Como exemplo, você pode usar o Yarn para gerenciar recursos do cluster, o Hive e o Spark para fornecer armazenamento off-line e computação de dados distribuídos em larga escala, o Spark Streaming e o Flink para oferecer computação de fluxo de dados, o Presto para permitir consultas interativas, e o Tez para fornecer uma estrutura de computação distribuída de gráficos acíclicos direcionados (DAGs).
  - Cluster de HBase: usa componentes Hadoop e HBase para fornecer um sistema de armazenamento distribuído em nuvem orientado a colunas que possua confiabilidade aprimorada, excelente desempenho e escalabilidade elástica. Aplica-se ao armazenamento e à computação distribuída de grandes quantidades de dados. Você pode utilizar o HBase para criar um sistema de armazenamento capaz de armazenar dados em nível de TB ou mesmo PB. Com o HBase, é possível filtrar e analisar dados com facilidade e obter respostas em milissegundos, explorando rapidamente o valor dos dados.
  - Cluster do Kafka: usa Kafka e Storm para fornecer um sistema de mensagens de código aberto com alta taxa de transferência e escalabilidade. É amplamente utilizado em casos de coleta de logs e de monitoramento da agregação de dados para implementar uma coleta eficiente de dados de streaming e o processamento e armazenamento de dados em tempo real.

Encerrar um cluster

O MRS permite que você encerrar um cluster quando ele não for mais necessário. Depois que o cluster for encerrado, todos os recursos de nuvem usados pelo cluster serão liberados. Antes de encerrar um cluster, é recomendável migrar ou fazer backup dos dados. Encerrar o cluster somente quando nenhum serviço estiver em execução no cluster ou o cluster estiver anormal e não puder fornecer serviços com base na análise de O&M. Se os dados forem armazenados em discos EVS ou discos de passagem em um cluster de Big Data, os dados serão excluídos após o término do cluster. Portanto, tenha cuidado ao encerrar um cluster.

7.9.2 Dimensionamento de cluster

A capacidade de processamento de um cluster de Big Data pode ser expandida horizontalmente adicionando nós. Se a escala de cluster não atender aos requisitos de serviço, você pode expandir manualmente ou reduzir no cluster. O MRS seleciona de forma inteligente o nó com a menor carga ou a quantidade mínima de dados a serem migrados para redução. O nó a ser reduzido não receberá novas tarefas e continuará a executar as tarefas existentes. Ao mesmo tempo, o MRS copia seus dados para outros nós e o nó é desativado. Se as tarefas no nó não puderem ser concluídas depois de muito tempo, o MRS migrará as tarefas para outros nós, minimizando o impacto nos serviços de cluster.

Expandir um cluster

Atualmente, você pode adicionar nós centrais ou de tarefa para dimensionar um cluster para lidar com cargas de serviço de pico. A adição de nós de cluster do MRS não afeta os serviços do cluster existente. Para obter detalhes sobre como corrigir a distorção de dados causada pela expansão da capacidade, consulte **Como o HDFS equilibra os dados?**.

Expandir um cluster cobrado no modo anual/mensal

Se a taxa de crescimento do serviço exceder o valor esperado após a assinatura de um cluster do MRS cobrado no modo **Yearly/Monthly**, é necessário expandir o cluster além da assinatura. O MRS permite-lhe expandir clusters cobrados no modo **Yearly/Monthly** enquanto desfruta dos descontos de subscrição.

Você pode acessar o console de gerenciamento do MRS e adicionar nós a um cluster com apenas alguns cliques. O processo de expansão do cluster não requer intervenção manual e leva apenas alguns minutos, o que ajuda a aliviar a pressão sobre as crescentes necessidades de processamento de dados de serviço.

Reduzir um cluster

Você pode reduzir o número de nós centrais ou de tarefa a serem dimensionados em um cluster para que o MRS ofereça melhores recursos de armazenamento e computação a menores custos de O&M com base nos requisitos de serviço. Depois de dimensionar em um cluster do MRS, o MRS seleciona automaticamente os nós que podem ser reduzidos com base no tipo de serviços instalados nos nós.

Durante a redução de nós centrais, os dados nos nós originais são migrados. Se a localização dos dados for armazenada em cache, o cliente atualizará automaticamente as informações de localização, o que poderá afetar a latência. A redução do nó pode afetar a duração da resposta do primeiro acesso a alguns dados do HBase em HDFS. Você pode reiniciar o HBase ou desabilitar ou ativar tabelas relacionadas para evitar esse problema.

Os nós de tarefa não armazenam dados de cluster. Eles são nós de computação e não envolvem migração de dados nos nós.

7.9.3 Auto Scaling

Introdução de recurso

Cada vez mais empresas usam tecnologias como Spark e Hive para analisar dados. O processamento de uma grande quantidade de dados consome enormes recursos e custa muito. Normalmente, as empresas regularmente analisam dados em um período fixo de tempo todos os dias, em vez do dia todo. Para atender aos requisitos das empresas, o MRS fornece a função de dimensionamento automático para solicitar recursos extras durante os horários de pico e liberar recursos fora dos horários de pico. Isso permite que os usuários usem recursos sob demanda e se concentrem no negócio principal a custos mais baixos.

Em aplicativos de Big Data, especialmente em cenários periódicos de análise e processamento de dados, os recursos de computação de cluster precisam ser ajustados dinamicamente com base nas alterações de dados de serviço para atender aos requisitos de serviço. A função de dimensionamento automático do MRS permite que os clusters sejam expandidos elasticamente ou reduzidos com base nas cargas do cluster. Além disso, se o volume de dados mudar regularmente e você quiser expandir para fora ou reduzir um cluster antes das alterações de volume de dados, você pode usar o recurso de plano de recursos MRS.

O MRS suporta dois tipos de políticas de dimensionamento automático: regras de dimensionamento automático e planos de recursos

- Regra de dimensionamento automático: você pode aumentar ou diminuir os nós de tarefa com base nas cargas de cluster em tempo real. O dimensionamento automático será acionado quando o volume de dados mudar, mas pode haver algum atraso.
- Planos de recursos: se o volume de dados mudar periodicamente, você poderá criar planos de recursos para redimensionar o cluster antes que o volume de dados seja alterado, evitando assim um atraso no aumento ou diminuição de recursos.

Tanto as regras de dimensionamento automático quanto os planos de recursos podem acionar dimensionamento automático. Você pode configurar ambas ou configurar uma delas. A configuração de planos de recursos e regras de dimensionamento automático melhora a escalabilidade do nó do cluster para lidar com picos de volume de dados ocasionalmente inesperados.

Em alguns cenários de serviço, os recursos precisam ser realocados ou a lógica de serviço precisa ser modificada após a expansão ou a ampliação do cluster. Se você expandir ou reduzir manualmente em um cluster, poderá efetuar logon nos nós do cluster para realocar recursos ou modificar a lógica do serviço. Se você usar o dimensionamento automático, o MRS permite que você personalize scripts de automação para realocação de recursos e modificação de lógica de serviço. Os scripts de automação podem ser executados antes e depois do dimensionamento automático e se adaptam automaticamente às alterações de carga de serviço, o que elimina as operações manuais. Além disso, os scripts de automação podem ser totalmente personalizados e executados em vários momentos, o que pode atender às suas necessidades personalizadas e melhorar a flexibilidade do dimensionamento automático.

Benefícios aos clientes

O dimensionamento automático do MRS oferece os seguintes benefícios:

- Reduzir custos

  As empresas não analisam dados o tempo todo, mas realizam uma análise de dados em lote em um período de tempo especificado, por exemplo, 03:00 da manhã. A análise do lote pode levar apenas duas horas.

  A função de dimensionamento automático permite que as empresas adicionem nós para análise em lote e liberem automaticamente os nós após a conclusão da análise, minimizando os custos.

- Atender aos requisitos de consulta instantânea

  As empresas geralmente encontram tarefas de análise instantânea, por exemplo, relatórios de dados para apoiar a tomada de decisões corporativas. Como resultado, o consumo de recursos aumenta acentuadamente em um curto período de tempo. Com a função de dimensionamento automático, os nós de computação podem ser adicionados para análise de Big Data emergente, evitando uma interrupção do serviço devido a recursos de computação insuficientes. Desta forma, você não precisa comprarcriar recursos extras. Depois que a emergência termina, o MRS libera automaticamente os nós.

- Focar no negócio principal

  É difícil para os desenvolvedores determinar o consumo de recursos na plataforma de desenvolvimento secundário de Big Data devido a condições complexas de análise de consulta (como classificação global, filtragem e mesclagem) e complexidade de dados, por exemplo, incerteza de dados incrementais. Como resultado, estimar o volume de

computação é difícil. A função de dimensionamento automático do MRS permite que os desenvolvedores se concentrem no desenvolvimento de serviços sem a necessidade de estimativa de recursos.

7.9.4 Criação de nó de tarefa

Introdução de característica

Os nós de tarefa podem ser criados e usados apenas para computação. Eles não armazenam dados persistentes e são a base para a implementação do dimensionamento automático.

Benefícios aos clientes

Quando o MRS é usado apenas como um recurso de computação, os nós de tarefas podem ser usados para reduzir custos e facilitar o dimensionamento dos nós do cluster, atendendo de forma flexível aos requisitos dos usuários para aumentar ou diminuir os recursos de computação do cluster.

Cenários de aplicação

Quando a alteração do volume de dados é pequena em um cluster, mas os recursos de processamento de serviço do cluster precisam ser notavelmente e temporariamente aprimorados, adicione nós de tarefa para resolver as seguintes situações:

- O número de serviços temporários é aumentado, por exemplo, o processamento de relatórios no final do ano.
- Tarefas de longo prazo precisam ser concluídas em um curto espaço de tempo, por exemplo, algumas tarefas de análise urgentes.

7.9.5 Ampliação de especificações do nó principal

O MRS fornece o Manager para gerenciar clusters e serviços nos clusters, como o NameNodes do HDFS, o ResourceManagers do Yarn e os serviços de gerenciamento Manager do MRS, implementados no nó principal dos clusters.

Com o lançamento de novos serviços, uma escala de cluster aumenta continuamente e os nós principais suportam cada vez mais cargas. Os usuários corporativos enfrentam o problema de que as cargas da CPU são muito altas e o uso de memória excede o limite. Geralmente, em um cluster de Big Data local, você precisa migrar dados e comprar hardware com configurações avançadas para amplia as especificações do nó principal. O MRS aproveita as vantagens dos serviços de nuvem para permitir que você amplia as especificações do nó principal em um clique. Durante a ampliação, o modo HA ativo/em espera dos nós principais garante que os serviços existentes não sejam interrompidos.

Para obter detalhes sobre como ampliar especificações de nó principal, consulte **Dimensionamento de especificações do nó principal**.

7.9.6 Isolamento de um host

Ao detectar que um host está anormal ou defeituoso e não pode fornecer serviços ou afetar o desempenho do cluster, você pode excluir o host dos nós disponíveis no cluster temporariamente para que o cliente possa acessar outros nós disponíveis. Em cenários em que os patches devem ser instalados em um cluster, você também pode excluir um nó especificado da instalação do patch. Somente os nós que não são de gerenciamento podem ser isolados.

Se um host for isolado, todas as instâncias de função no host serão paradas, e você não poderá iniciar, parar ou configurar o host e todas as instâncias no host. Além disso, depois que um host é isolado, as estatísticas sobre o status de monitoramento e os dados métricos de hardware e instâncias no host não podem ser coletadas ou exibidas.

7.9.7 Gerenciamento de tags

Tags são identificadores de cluster. Adicionar tags a clusters pode ajudá-lo a identificar e gerenciar seus recursos de cluster. Ao associar-se ao Tag Management Service (TMS), o MRS permite que os usuários com um grande número de recursos em nuvem marquem recursos em nuvem, pesquisem rapidamente recursos de nuvem com o mesmo atributo de tag e executem operações de gerenciamento unificadas, como revisão, modificação e exclusão, facilitando o gerenciamento unificado de clusters de Big Data e outros recursos em nuvem.

Você pode adicionar no máximo 10 tags a um cluster ao criar o cluster ou adicioná-las na página de detalhes do cluster criado.

7.10 O&M de cluster

Gestão de alarmes

O MRS pode monitorar clusters de Big Data em tempo real e identificar o status de integridade do sistema com base em alarmes e eventos. Além disso, o MRS permite personalizar o monitoramento e os limites de alarme para se concentrar no status de saúde de cada métrica. Quando os dados de monitoramento atingem o limite de alarme, o sistema dispara um alarme.

O MRS também pode se interconectar com o sistema de serviço de mensagens do serviço Simple Message Notification (SMN) da Huawei Cloud para enviar informações de alarme aos usuários por mensagem SMS ou e-mail. Para mais detalhes, consulte **Notificação de mensagem**.

Gerenciamento de patches

O MRS suporta operações de patching de cluster e liberará patches para componentes de Big Data de código aberto em tempo hábil. Na página de gerenciamento de cluster do MRS, você pode exibir as informações da versão de patch relacionadas aos clusters em execução, incluindo a descrição detalhada dos problemas e impactos resolvidos. Você pode determinar se deseja instalar um patch com base no status do serviço em execução. A instalação de patch com um clique não envolve intervenção manual e não causará interrupção do serviço por meio da instalação contínua, garantindo a disponibilidade a longo prazo dos clusters.

O MRS pode exibir o processo detalhado de instalação do patch. O gerenciamento de patches também oferece suporte à desinstalação e reversão de patches.

NOTA

O MRS 3.x ou posterior não oferece suporte ao gerenciamento de patches no console de gerenciamento.

Suporte de O&M

Os recursos de cluster fornecidos pelo MRS pertencem aos usuários. Geralmente, quando o suporte da equipe de O&M é necessário para a solução de problemas de um cluster, a equipe

de O&M não pode acessar diretamente o cluster. Para melhor atender os clientes, a MRS oferece os dois métodos a seguir para melhorar a eficiência da comunicação durante a localização de falhas:

- Compartilhamento de logs: você também pode iniciar o compartilhamento de logs no console de gerenciamento do MRS para compartilhar um escopo de log especificado com a equipe de O&M, de modo que a equipe de O&M possa localizar falhas sem acessar o cluster.
- Autorização de O&M: se ocorrer um problema quando você usar um cluster de MRS, poderá iniciar a autorização de O&M no console de gerenciamento de MRS. A equipe de O&M pode ajudá-lo a localizar rapidamente o problema e você pode revogar a autorização a qualquer momento.

Verificação de integridade

O MRS fornece inspeção automática em ambientes em execução do sistema para que você possa verificar e auditar o status de integridade do sistema em execução em um clique, garantindo o funcionamento adequado do sistema e reduzindo os custos de operação e manutenção do sistema. Depois de visualizar os resultados da inspeção, você pode exportar relatórios para arquivamento e análise de falhas.

7.11 Notificação de mensagem

Introdução de recursos

As seguintes operações são frequentemente realizadas durante a execução de um cluster de Big Data:

- Os clusters de Big Data geralmente mudam, por exemplo, a expansão e redução do cluster.
- Quando um volume de dados de serviço muda abruptamente, o escalonamento automático será acionado.
- Depois que os serviços relacionados são interrompidos, um cluster de Big Data precisa ser interrompido.

Para notificá-lo imediatamente de operações bem-sucedidas, indisponibilidade de cluster e falhas de nó, o MRS usa Simple Message Notification (SMN) para enviar notificações por SMS e e-mails, facilitando a manutenção.

Benefícios aos clientes

Depois de configurar o SMN, você pode receber o status de integridade do cluster do MRS, atualizações e alarmes de componentes por meio de SMS ou e-mails em tempo real. O MRS envia monitoramento em tempo real e notificação de alarme para ajudá-lo a executar facilmente O&M e implementar eficientemente serviços de Big Data.

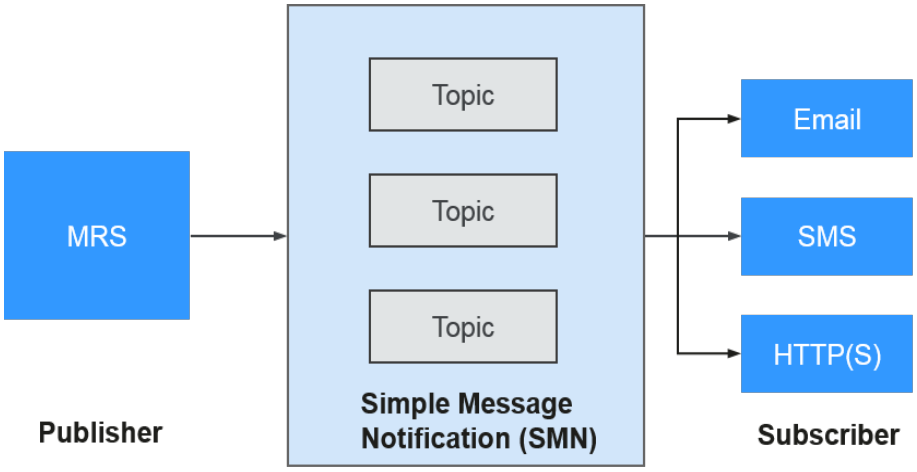
Descrição de recursos

MRS usa SMN para fornecer assinatura e notificação de mensagem um-para-múltiplos em uma variedade de protocolos.

Você pode criar um tópico e configurar políticas de tópico para controlar as permissões de editor e assinante no tópico. O MRS envia mensagens de cluster para o tópico no qual você

tem permissão para publicar mensagens. Em seguida, todos os assinantes que assinam o tópico podem receber atualizações de cluster e alarmes de componentes por meio de SMS e e-mails.

Figura 7-1 Processo de implementação



8 Segurança

8.1 Responsabilidades compartilhadas

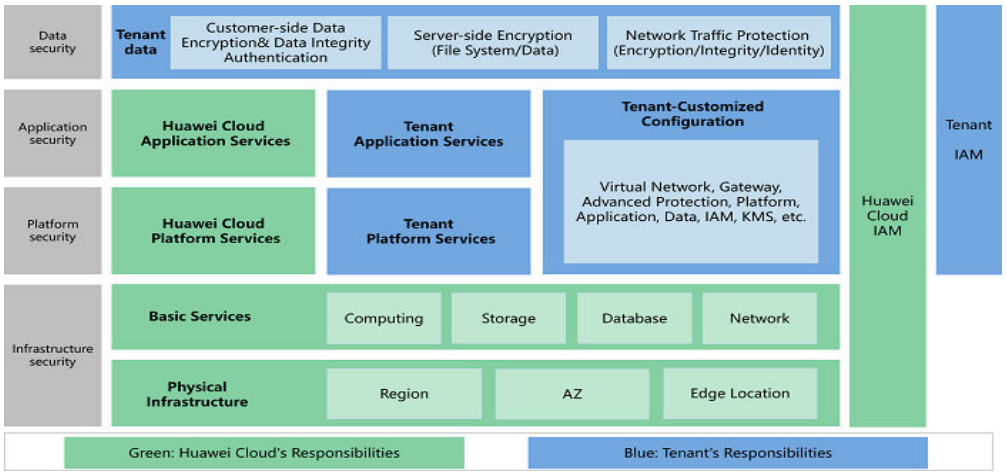
Huawei Cloud garante que seu compromisso com a segurança cibernética nunca será superado pela consideração de interesses comerciais. Para lidar com os desafios emergentes de segurança na nuvem e ameaças e ataques à segurança na nuvem, a Huawei Cloud constrói um sistema abrangente de garantia de segurança de serviços em nuvem para diferentes regiões e indústrias com base nas vantagens exclusivas de software e hardware, leis, regulamentos, padrões da indústria e ecossistema de segurança da Huawei.

Figura 8-1 ilustra as responsabilidades partilhadas pela Huawei Cloud e pelos usuários.

- Huawei Cloud: garante a segurança dos serviços em nuvem e fornece nuvens seguras. As responsabilidades de segurança da Huawei Cloud incluem garantir a segurança de nossos serviços de IaaS, PaaS e SaaS, bem como os ambientes físicos dos data centers da Huawei Cloud onde nossos serviços de IaaS, PaaS e SaaS operam. A Huawei Cloud é responsável não apenas pelas funções de segurança e pelo desempenho de nossa infraestrutura, serviços de nuvem e tecnologias, mas também pela segurança geral de O&M na nuvem e, no sentido mais amplo, pela conformidade de segurança de nossa infraestrutura e serviços.
- Locatário: usa a nuvem com segurança. Os locatários da Huawei Cloud são responsáveis pelo gerenciamento seguro e eficaz da segurança interna, bem como pelas configurações personalizadas dos serviços de nuvem, incluindo IaaS, PaaS e SaaS. Isso inclui, mas não se limita a, sistemas operacionais como redes virtuais, host de máquina virtual e máquinas virtuais convidadas, firewall virtual, API Gateway e serviços avançados de segurança, todos os tipos de serviços de nuvem, dados de locatários, contas de identidade e gerenciamento de chaves.

Livro branco de segurança da Huawei Cloud apresenta em detalhes as ideias e medidas de construção da segurança em nuvem da Huawei, incluindo estratégia de segurança em nuvem, modelo de compartilhamento de responsabilidade, conformidade e privacidade, organização e pessoal de segurança, segurança de infraestrutura, serviço de locatário e segurança de inquilino, segurança de engenharia, O&M e segurança de operação e segurança do ecossistema.

Figura 8-1 Modelo de responsabilidade de segurança compartilhada da Huawei Cloud



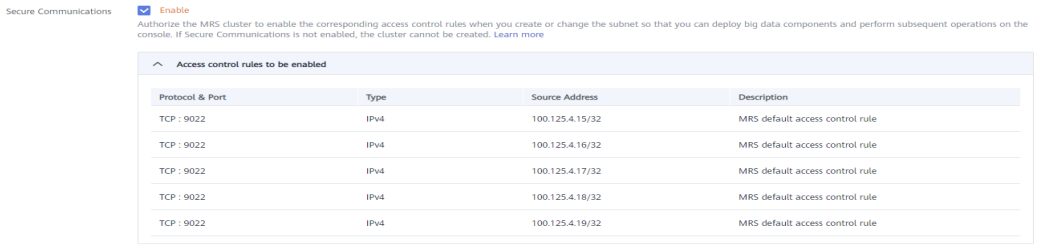
8.2 Identificação e gerenciamento de ativos

Comunicações seguras

Em um cluster do MRS, você pode provisionar, gerenciar e usar componentes de Big Data por meio do console de gerenciamento. Os componentes de Big Data são implementados nas VPCs dos usuários. Para permitir que o console do MRS acesse diretamente os componentes de Big Data, você deve habilitar as regras de grupo de segurança correspondentes após conceder a autorização. Esse processo de autorização é chamado de comunicações seguras.

Você precisa habilitar comunicações seguras ao criar um cluster, conforme mostrado na **Figura 8-2**.

Figura 8-2 Comunicações seguras

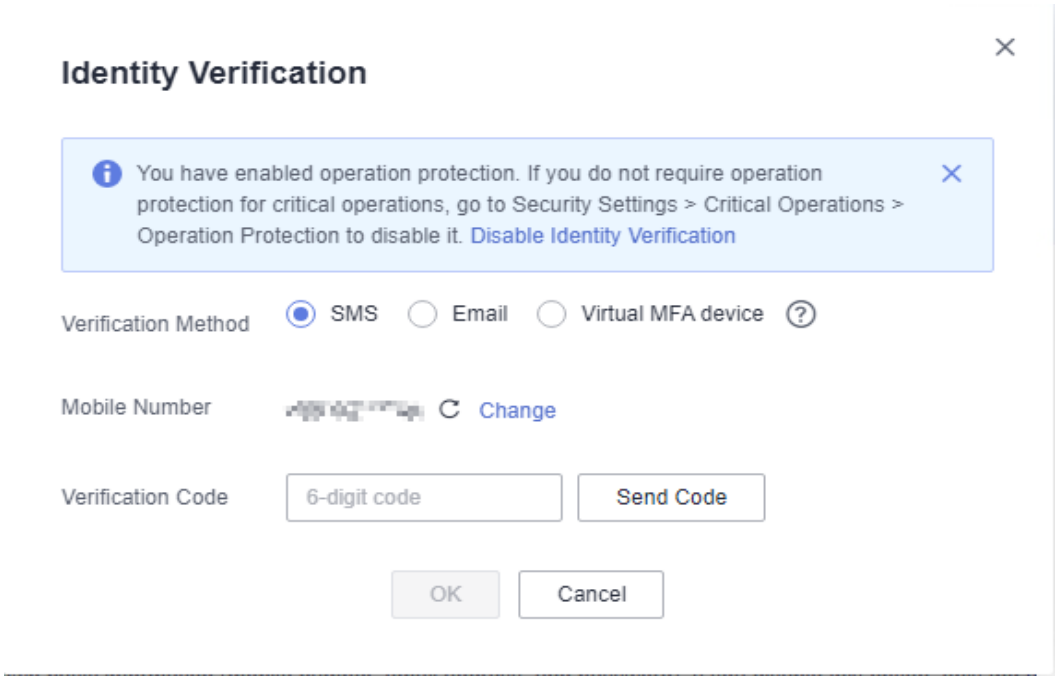


Recomendamos que você habilite as regras de grupo de segurança somente para endereços IP confiáveis. Tenha cuidado ao usar 0.0.0.0/0 como endereço de origem do grupo de segurança.

Proteção para operações críticas

O MRS fornece proteção para operações críticas. Se você ativou a proteção de operação (para obter detalhes, consulte **Proteção de operação crítica** do IAM), insira o código de verificação obtido usando o método de verificação selecionado (conforme mostrado em **Figura 8-3**) para evitar riscos e perdas causados por operações incorretas.

Figura 8-3 Verificação de identidade



8.3 Autenticação de identidade e controle de acesso

Autenticação de identificação

O MRS oferece suporte ao protocolo de segurança Kerberos. O FusionInsight MRS emprega LDAP para o sistema de gerenciamento de contas e realiza autenticação de segurança nas informações da conta via Kerberos.

Para obter detalhes sobre o mecanismo de autenticação de segurança Kerberos, consulte **Princípios e mecanismos de autenticação de segurança**.

Controle de acesso

O MRS fornece dois modelos de controle de acesso: controle de acesso baseado em função e controle de acesso baseado em política. Para obter detalhes, consulte **Modelo de direito**.

- **Controle de acesso baseado em função**

  Empregando um sistema unificado de autenticação baseado em usuário e função e em conformidade com o modelo de controle de acesso baseado em conta/função (RBAC), o MRS implementa o gerenciamento de permissões baseado em função e o gerenciamento de autorização de usuário em lote. Ele também fornece o recurso de logon único (SSO) para oferecer gerenciamento e autenticação unificados para usuários do sistema do FusionInsight MRS e usuários de componentes. Para detalhes sobre o mecanismo, veja **Mecanismo de direito**.

- **Controle de acesso baseado em políticas**
  - Autenticação Ranger

    O MRS suporta a autenticação Ranger. Para um cluster de MRS no modo de segurança, a autenticação Ranger é ativada por padrão. Para um cluster normal com

o serviço Ranger instalado, o Ranger suporta controle de permissão em recursos de componentes com base em usuários do sistema operacional.

Para obter detalhes sobre as políticas de autenticação do Ranger, consulte **Políticas de verificação de permissão**.

- Autenticação fina para clusters desacoplados de computação de armazenamento do OBS

  Se você deseja executar um controle de permissão refinado em recursos do OBS em clusters desacoplados de computação de armazenamento do OBS, o MRS fornece uma solução de controle de permissão refinada baseada na agência do IAM.

  Para obter detalhes, consulte **Configuração de permissões refinadas para acesso multiusuário do MRS ao OBS**.

8.4 Tecnologias de proteção de dados

Integridade dos dados

Os dados são verificados para garantir sua integridade durante o armazenamento e a transmissão.

Os dados do usuário do MRS são armazenados no HDFS, que usa o CRC32C para verificar os dados. O HDFS também suporta a verificação CRC32, que é muito mais rápida que o CRC32C. Os DataNodes de HDFS armazenam os dados verificados. Se detectar que os dados transmitidos do cliente estão incompletos, eles relatam a exceção ao cliente e notificam o cliente sobre a retransmissão de dados. O cliente verifica a integridade dos dados ao ler dados de um DataNode. Se os dados estiverem incompletos, o cliente lerá os dados de outro DataNode.

Confidencialidade de dados

Baseado no Apache Hadoop, o sistema de arquivos distribuído do FusionInsight MRS fornece armazenamento criptografado de arquivos para evitar que dados confidenciais sejam armazenados em texto simples, melhorando a segurança dos dados.

As aplicações precisam criptografar apenas dados confidenciais especificados. Os serviços não são afetados durante a criptografia e descriptografia. Além da criptografia de dados do sistema de arquivos, o Hive fornece criptografia de coluna (consulte **Uso da função de criptografia de coluna do Hive**). Os dados confidenciais podem ser criptografados e armazenados após você especificar um algoritmo de criptografia durante a criação da tabela. O HBase suporta criptografia de HFiles e WALs (consulte **Criptografia de HFile e WAL**). Você pode configurar os algoritmos AES e SMS4 para criptografá-los.

Segurança de transmissão de dados

Em um cluster de MRS, a criptografia HTTPS é suportada para acesso por canais da Web. A comunicação RPC suporta autenticação SASL e suporta criptografia de dados usando chaves simétricas. A configuração de transmissão criptografada de cada componente é a seguinte:

- Configuração de transmissão criptografada HDFS: consulte **Configuração da criptografia de dados do HDFS durante a transmissão**.
- Configuração de transmissão encriptada do Kafka: consulte **Configuração da criptografia de dados do Kafka durante a transmissão**.

- Configuração de transmissão criptografada pelo Flume: consulte **Configuração da Transmissão Criptografada**.
- Configuração de transmissão encriptada pelo Flink: consulte **Transmissão criptografada** em **Autenticação e Criptografia**.

Backup de dados e recuperação de desastres

- Recuperação de desastres (DR): o MRS suporta backup de dados para o OBS da Huawei Cloud e oferece alta confiabilidade entre regiões.
- Backup: o FusionInsight MRS pode fazer backup dos metadados do OMS, Kafka, DBService e NameNodes bem como dos dados de serviço do HDFS, HBase e Hive.

Para obter detalhes, consulte **Introdução ao gerenciamento de backup e recuperação**.

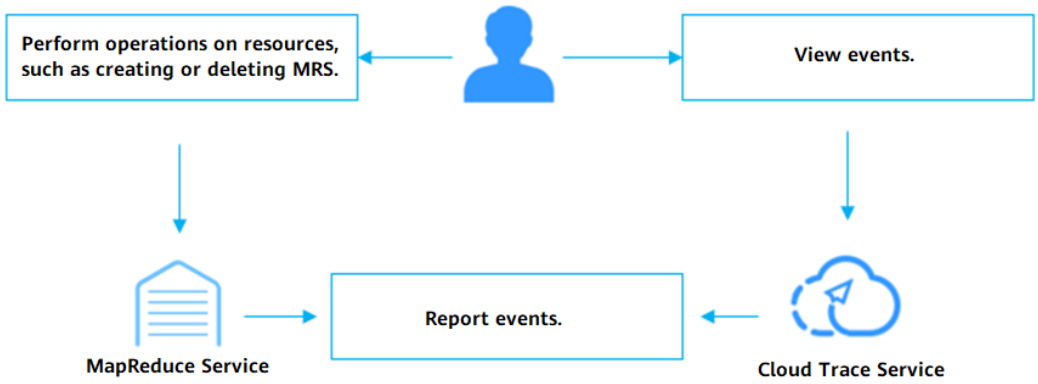
8.5 Auditoria e registro em log

Auditoria

O Cloud Trace Service (CTS) registra os logs de operação do MRS no console de gerenciamento, como criar ou excluir clusters do MRS. O CTS é um serviço de auditoria de logs destinado à segurança na nuvem. Ele registra as operações nos recursos da nuvem em sua conta. Você pode usar os logs gerados pelo CTS para realizar análises de segurança, rastrear alterações de recursos, auditar conformidade e localizar falhas.

Depois de ativar o CTS e configurar um rastreador, o CTS pode registrar gerenciamento e rastros de dados do MRS para auditoria.

Figura 8-4 Gravação de eventos do MRS no CTS



O FusionInsight Manager fornece a função de auditoria para registrar as operações do usuário no Manager de clusters. Na página **Auditoria**, os administradores podem exibir os registros históricos de operação dos usuários no Manager. Os registros podem ser usados para localizar falhas e determinar responsabilidades em eventos de segurança. Para obter detalhes sobre a página **Auditoria**, consulte **Visão geral da auditoria**. Os logs de auditoria do FusionInsight Manager são armazenados no banco de dados por padrão. Se os logs de auditoria forem retidos por um longo período de tempo, o espaço em disco do diretório de dados pode se tornar insuficiente. Para armazenar logs de auditoria em outro servidor de arquivamento, os administradores podem definir os parâmetros de despejo necessários para despejar

automaticamente esses logs. Isso facilita o gerenciamento de logs de auditoria. Para obter detalhes sobre como despejar logs de auditoria, consulte **Configuração do despejo de log de auditoria**.

Registro em log

Os logs de todos os componentes em um cluster do MRS (por exemplo, todos os logs HDFS) podem ser coletados conectando hosts ao Log Tank Service (LTS). O LTS coleta dados de log de hosts e serviços em nuvem. Ao analisar e processar grandes quantidades de logs de forma eficiente, segura e em tempo real, o LTS fornece informações úteis para você otimizar a disponibilidade e o desempenho de serviços e aplicativos em nuvem. Ele também ajuda você a executar com eficiência a tomada de decisões em tempo real, o gerenciamento de O&M de dispositivos e a análise de tendências de serviços. Para obter detalhes sobre a interconexão, consulte **Como interconectar o MRS com o LTS**.

FusionInsight Manager também suporta a pesquisa on-line de logs de componentes para localização de falhas e outros cenários. Para obter detalhes, consulte **Pesquisa de log on-line**. Além disso, o FusionInsight Manager permite exportar logs gerados por todas as instâncias de cada função de serviço em lotes, sem a necessidade de efetuar logon manualmente em cada nó. Para obter detalhes, consulte **Download de log**.

8.6 Resiliência de serviço

Implementação de DR entre AZs

O plano de gestão de MRS fornece a capacidade DR de cluster duplo entre AZs. Você pode implantar um cluster homogêneo de DR do MRS em outra AZ. Se o cluster de produção não fornecer serviços de leitura e gravação devido a um desastre natural ou falhas internas do cluster, o cluster de DR se tornará o cluster de produção para garantir a continuidade do serviço.

8.7 Monitoramento de riscos de segurança

Manager do serviço MRS fornece recursos de monitoramento em nível de cluster para ajudá-lo a monitorar o status de integridade de componentes e nós de big data em clusters. Ele também fornece o recurso de notificação de alarme para que você possa aprender sobre as métricas e o status de integridade dos clusters de MRS em tempo real.

8.8 Gerenciamento de atualização

Atualização de senha

O MRS suporta atualização de senha. É aconselhável alterar a senha periodicamente de acordo com as seguintes orientações para melhorar a segurança do sistema:

- Alterar a senha de um usuário do sistema: consulte **Alteração da senha do usuário admin** e **Alteração da senha de um usuário do sistema operacional**.
- **Alteração da senha de um usuário interno do sistema**
- **Alteração da senha de um usuário de banco de dados**

Atualização de certificado

Tanto o certificado CA quanto o certificado HA do MRS podem ser substituídos. Você pode substituir os certificados padrão de um cluster de acordo com a seguinte orientação:

- O certificado CA do MRS é usado para criptografar dados durante a comunicação entre o cliente e o servidor de um componente para garantir a segurança da comunicação. Para obter detalhes sobre como substituir um certificado AC, consulte **Substituição do certificado AC**.
- O certificado HA é usado para criptografar dados durante a comunicação entre processos ativos/em espera e processos HA para garantir a segurança da comunicação. Para obter detalhes sobre como substituir um certificado HA, consulte **Substituição de certificados HA**.

8.9 Fortalecimento de segurança

Fortalecimento de Tomcat

Durante a instalação e o uso do FusionInsight Manager, as seguintes funções do Tomcat são aprimoradas com base na versão de código aberto:

- Tomcat é atualizado para uma versão oficial estável.
- As permissões nos diretórios sob aplicativos são definidas como **500**, e a permissão de gravação em alguns diretórios é suportada.
- O pacote de instalação do Tomcat é automaticamente excluído após a instalação do software do sistema.
- A função de implementação automática está desativada para projetos em diretórios de aplicações. Somente os projetos **web**, **cas** e **client** são implementados.
- Alguns métodos **http** não utilizados estão desabilitados, impedindo ataques que podem ser lançados usando os métodos **http**.
- A porta de desligamento padrão e o comando do servidor de Tomcat são alterados para evitar que hackers desliguem o servidor e ataquem o servidor e as aplicações.
- Para garantir a segurança, o valor de **maxHttpHeaderSize** é alterado, o que permite que os administradores do servidor controlem solicitações anormais de clientes.
- O arquivo de descrição da versão do Tomcat é modificado após a instalação do Tomcat.
- Para evitar a divulgação de informações do Tomcat, os atributos do Servidor do Connector são modificados para que os invasores não possam obter informações sobre o servidor.
- Permissões em arquivos e diretórios do Tomcat, como arquivos de configuração, arquivos executáveis, diretórios de log e pastas temporárias, estão sob controle.
- A reciclagem de fachadas de sessão está desativada para evitar vazamentos de solicitações.
- LegacyCookieProcessor é usado como CookieProcessor para evitar o vazamento de dados confidenciais nos cookies.

Fortalecimento de LDAP

O LDAP é reforçado da seguinte forma após a instalação de um cluster:

- No arquivo de configuração de LDAP, a senha da conta de administrador é criptografada usando SHA. Depois que o OpenLDAP é atualizado para 2.4.39 ou posterior, os dados são sincronizados automaticamente entre os nós de LDAP ativos e em espera usando o Mecanismo externo SASL, o que impede a divulgação da senha.
- O serviço LDAP no cluster suporta o protocolo SSLv3 por padrão, que pode ser usado com segurança. Quando o OpenLDAP é atualizado para 2.4.39 ou posterior, o LDAP usa automaticamente o TLS1.0 ou posterior para evitar riscos de segurança desconhecidos.

Outro fortalecimento de segurança

Para obter detalhes sobre outras orientações de proteção de segurança, consulte **Fortalecimento de segurança**.

9 Restrições

Antes de usar o MRS, certifique-se de ter lido e entendido as seguintes restrições.

- Os clusters do MRS devem ser criados em sub-redes da VPC.
- É aconselhável usar qualquer um dos seguintes navegadores para acessar o MRS:
  - Google Chrome: 36.0 ou mais recente
  - Internet Explorer: 9.0 ou mais recente

    Se utilizar o Internet Explorer 9.0, poderá não iniciar sessão na consola de gestão MRS porque o usuário **Administrator** está desativado por predefinição em alguns sistemas Windows, como o Windows 7 Ultimate. O Internet Explorer seleciona automaticamente um usuário do sistema para instalação. Como resultado, o Internet Explorer não pode acessar o console de gerenciamento. Reinstale o Internet Explorer 9.0 ou posterior (recomendado) ou execute o Internet Explorer 9.0 como usuário **Administrator**.
- Ao criar um cluster de MRS, você pode selecionar **Auto create** na lista suspensa de **Security Group** para criar um grupo de segurança ou selecionar um grupo de segurança existente. Depois que o cluster do MRS for criado, não exclua ou modifique o grupo de segurança usado. Caso contrário, uma exceção de cluster pode ocorrer.
- Para evitar o acesso ilegal, atribua apenas permissão de acesso para grupos de segurança usados pelo MRS quando necessário.
- Não execute as seguintes operações porque causarão exceções de cluster:
  - Desligar, reiniciar ou excluir nós de cluster MRS exibidos no ECS, alterar ou reinstalar seu sistema operacional ou modificar suas especificações.
  - Excluir os processos, aplicações ou arquivos existentes em nós de cluster.
  - Excluir nós de cluster do MRS. Os nós excluídos ainda serão cobrados.
- Se ocorrer uma exceção de cluster quando nenhuma operação incorreta tiver sido realizada, entre em contato com os engenheiros de suporte técnico. Eles vão pedir-lhe para a sua senha e, em seguida, executar a solução de problemas.
- Mantenha a senha inicial para fazer logon no nó mestre corretamente porque o MRS não a salvará. Use uma senha complexa para evitar ataques maliciosos.
- Os clusters do MRS ainda são cobrados durante as exceções. Entre em contato com engenheiros de suporte técnico para lidar com exceções de cluster.
- Planeje discos de nós de cluster com base nos requisitos de serviço. Se você quiser armazenar um grande volume de dados de serviço, adicione discos EVS ou espaço de

armazenamento para evitar que o espaço de armazenamento insuficiente afete a execução do nó.

- Os nós do cluster armazenam apenas os dados de serviço dos usuários. Os dados que não são de serviço podem ser armazenados no OBS ou em outros nós do ECS.
- Os nós de cluster executam apenas programas de cluster do MRS. Outras aplicações de cliente ou programas de serviço ao usuário são implementados em nós ECS separados.
- Para expandir a capacidade de armazenamento dos nós (incluindo mestre, núcleo e tarefa) em um cluster MRS, comprecrie novos discos e, em seguida, conecte-os aos nós. Para obter detalhes, consulte **Expansão da capacidade de um disco EVS em uso**.
- A capacidade (incluindo capacidades de armazenamento e computação) de um cluster do MRS pode ser expandida pela adição de nós central ou de tarefa.
- Se o cluster ainda for usado para executar tarefas ou modificar configurações depois que um nó principal no cluster for interrompido e outros nós principais no cluster forem interrompidos antes que o nó principal interrompido seja iniciado após a execução da tarefa ou a modificação da configuração, dados podem ser perdidos devido a uma alternância ativa/em espera. Nesse cenário, depois que a tarefa é executada ou a configuração é modificada, inicie o nó principal que foi interrompido e, em seguida, pare todos os nós. Se todos os nós no cluster tiverem sido interrompidos, inicie-os na ordem inversa de desligamento do nó.
- A alternância do agendador da capacidade e superior é concluída quando o cluster de MRS é usado, enquanto a sincronização de configuração não é concluída. Configure a sincronização novamente com base no novo agendador, se necessário.

10 Cobrança

O preço do MapReduce Service (MRS) é simples e previsível. Você pode selecionar um modo de cobrança de pagamento por uso ou assinatura anual/mensal, dependendo do que é mais econômico para você. O preço total de um cluster do MRS será calculado automaticamente para que você possa comprar um cluster com um clique.

Itens cobrados

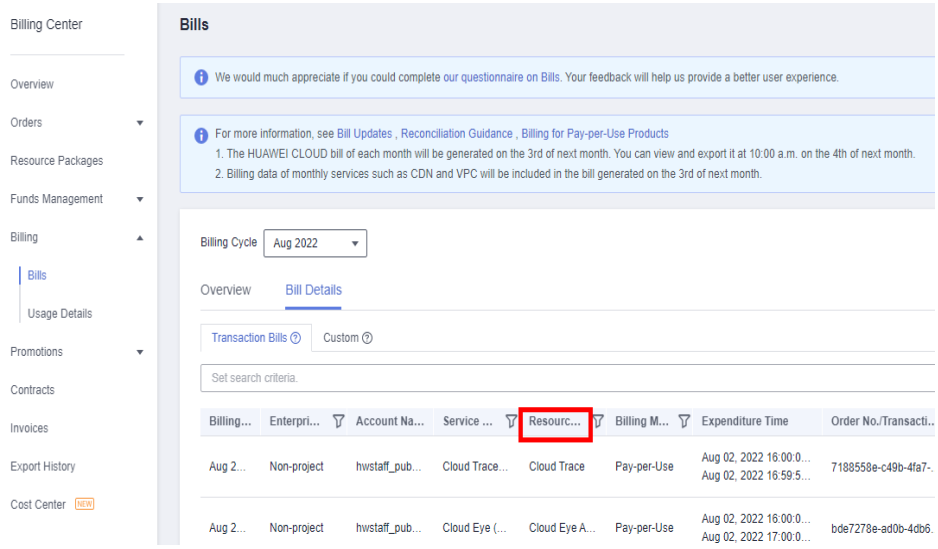
O preço de um cluster do MRS consiste em duas partes:

- Taxa de gerenciamento do MRS

NOTA

Você pode visualizar a taxa de gerenciamento de MRS detalhada fazendo logon no Central de cobrança, escolhendo **Billing** > **Bills** e filtrando as taxas de gerenciamento.

Figura 10-1 Visualizar a taxa de gerenciamento do MRS



- Se o **Cluster Type** for **LTS**, filtre as taxas de gerenciamento por **MRS-LTS Service Fee**.
- Se o **Cluster Type** for **Normal**, filtre as taxas de gerenciamento por:
  - **MapReduce Service VM** para clusters adquiridos em junho de 2022 ou anterior
  - **MRS-BASIC Service Fee** para clusters adquiridos após junho de 2022

- Taxas de recursos de infraestrutura IaaS, incluindo Elastic Cloud Server (ECS), Elastic Volume Service (EVS), Elastic IP (EIP) e largura de banda

O cluster do MRS encerrado ou cancelado não é mais cobrado.

Modo de cobrança

Antes de usar o MRS, você deve adquirir um cluster do MRS. Atualmente, o MRS oferece dois modos de cobrança:

- Anual/mensal: você pode pagar por clusters por ano ou mês. A duração mínima é de 1 mês e a duração máxima é de 1 ano.
- Pagamento por uso: os nós são cobrados pela duração real do uso, com um ciclo de cobrança de uma hora.

Alteração do modo de cobrança

Antes de assinar o MRS, escolha as instâncias de nó principal e central que melhor atendam às suas necessidades. O MRS fornece os seguintes métodos para você alterar a configuração do cluster depois que um cluster é iniciado.

- **Configurar nó de tarefa**: adicionar nós de tarefa. Para obter detalhes, consulte **Operações relacionadas** em **Expansão manual de um cluster**.
- **Expandir**: adicionar manualmente os nós central ou de tarefa. Para obter detalhes, consulte **Expansão manual de um cluster**.

- **Dimensionamento automático**: o número de nós em um cluster pode ser ajustado automaticamente com base no volume de dados do serviço para aumentar ou diminuir os recursos. Para obter detalhes, consulte **Configuração de regras do dimensionamento automático**.

Se os métodos de alteração de configuração fornecidos pelos MRS não atender aos seus requisitos, você poderá criar um cluster novamente e migrar os dados para o cluster para realizar a alteração na configuração do cluster.

Renovação

Para renovar a assinatura, acesse a página **Renovações**.

Pagamento em atraso

O pagamento em atraso não se aplica a clusters subscritos anualmente ou mensalmente.

No modo de pagamento por uso, as taxas de cluster são deduzidas a cada hora. Se o saldo da sua conta for insuficiente para pagar a despesa ocorrida na última hora, sua conta ficará em atraso e os clusters de MRS terão um **período de retenção**. Se os clusters forem renovados dentro do período de retenção, eles estarão disponíveis e serão cobrados a partir da data de expiração original.

É aconselhável renovar o cluster o mais rápido possível se o cluster estiver em atraso. Caso contrário, as seguintes operações são restringidas:

- Criar clusters
- Expandir um cluster
- Reduzir um cluster
- Adicionar um nó de Tarefa
- Ampliar especificações de nó Master

Expiração

- A expiração não se aplica a clusters de pagamento por uso.
- Se sua assinatura anual ou mensal expirar, o cluster entrará em um **período de retenção**. Durante o período de carência e o período de retenção, não é possível executar operações no cluster no console de gerenciamento do MRS, as APIs relacionadas não podem ser chamadas e as operações de O&M, como monitoramento automático e emissão de relatórios de alarmes, serão interrompidas. Se sua assinatura não for renovada no final do período de retenção, os serviços no cluster serão encerrados e os dados no sistema serão excluídos permanentemente.

11 Gerenciamento de permissões

Se você precisar atribuir permissões diferentes a funcionários em sua empresa para acessar seus recursos de MRS na Huawei Cloud, o IAM é uma boa escolha para o gerenciamento de permissões refinado. O IAM fornece autenticação de identidade, gerenciamento de permissões e controle de acesso, ajudando você a proteger o acesso aos seus recursos da Huawei Cloud.

Com o IAM, você pode criar usuários do IAM em sua conta da Huawei Cloud e atribuir permissões a esses usuários para controlar seu acesso a recursos específicos. Por exemplo, alguns desenvolvedores de software em sua empresa precisam usar recursos do MRS, mas não devem excluir os clusters ou executar operações de alto risco. Para alcançar esse resultado, você pode criar usuários do IAM para os desenvolvedores de software e conceder a eles apenas as permissões necessárias para usar os recursos de cluster do MRS.

Se sua conta da Huawei Cloud não exigir usuários individuais do IAM para gerenciamento de permissões, pule esta seção.

O IAM é gratuito. Você paga apenas pelos recursos que usa. Para obter mais informações sobre o IAM, consulte **Visão geral de serviço do IAM**.

Descrição da permissão de MRS

Por padrão, novos usuários do IAM não têm nenhumas permissões concedidas. Para atribuir permissões a um usuário, adicione o usuário a um ou mais grupos e atribua políticas de permissões ou funções a esses grupos. O usuário então herda as permissões dos grupos dos quais é membro e pode executar operações especificadas em serviços de nuvem com base nas permissões.

O MRS é um serviço de nível de projeto implementado e acessado em regiões físicas específicas. Para atribuir permissões a um grupo de usuários, especifique **Scope** como **Region-specific projects** e selecione projetos na região correspondente para que as permissões entrem em vigor. Se **All projects** estiver selecionado, as permissões entrarão em vigor para o grupo de usuários em todos os projetos específicos da região. Ao acessar o MRS, os usuários precisam mudar para uma região onde foram autorizados a usar o serviço MRS.

Você pode conceder permissões aos usuários usando funções e políticas.

- Funções: um tipo de mecanismo de autorização de granulação grosseira que define permissões relacionadas às responsabilidades do usuário. Esse mecanismo fornece apenas um número limitado de funções de nível de serviço para autorização. Ao usar funções para conceder permissões, você também precisa atribuir outras funções das quais

as permissões dependem para entrar em vigor. No entanto, as funções não são uma escolha adequada para autorização refinada e controle de acesso seguro.

- Políticas: um tipo de mecanismo de autorização refinado que define as permissões necessárias para realizar operações em recursos em nuvem específicos sob determinadas condições. Esse mecanismo permite uma autorização baseada em políticas mais flexível, atendendo aos requisitos de controle de acesso seguro. Por exemplo, você pode conceder aos usuários do MRS somente as permissões para executar operações especificadas em clusters do MRS, como criar um cluster e consultar uma lista de clusters em vez de excluir um cluster. A maioria das políticas define permissões com base em APIs. Para as ações de API suportadas pelo MRS, consulte **Políticas de permissões e ações suportadas**.

Tabela 11-1 lista todas as políticas do sistema suportadas pelo MRS.

Tabela 11-1 Políticas do sistema do MRS

Política	Descrição	Tipo
MRS FullAccess	Permissões de administrador para MRS. Os usuários com essas permissões podem operar e usar todos os recursos do MRS.	Política refinada
MRS CommonOperations	Permissões de usuário comuns para MRS. Os usuários concedidos a essas permissões podem usar o MRS, mas não podem adicionar ou excluir recursos.	Política refinada
MRS ReadOnlyAccess	Permissões somente leitura para o MRS. Os usuários com essas permissões só podem visualizar os recursos do MRS.	Política refinada
MRS Administrator	Permissões: ● Todas as operações no MRS ● Os usuários com permissões desta política também devem receber permissões das políticas **Tenant Guest** e **Server Administrator**.	Política de RBAC

Tabela 11-2 lista as operações comuns suportadas por cada política ou função definida pelo sistema do MRS. Selecione as políticas ou funções conforme necessário.

Tabela 11-2 Operações comuns suportadas por cada política definida pelo sistema

Operação	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Criar um cluster	√	x	x	√

Operação	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Redimensionar um cluster	√	x	x	√
Atualizar especificações de nó	√	x	x	√
Excluir um cluster	√	x	x	√
Consultar detalhes de cluster	√	√	√	√
Consultar lista de clusters	√	√	√	√
Configurar uma regra de dimensionamento automático	√	x	x	√
Consultar uma lista de hosts	√	√	√	√
Consultar logs de operação	√	√	√	√
Criar e executar um job	√	√	x	√
Parar um job	√	√	x	√
Excluir um único job	√	√	x	√
Excluir jobs em lotes	√	√	x	√
Consultar detalhes do job	√	√	√	√

Operação	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Consultar uma lista de jobs	√	√	√	√
Criar uma pasta	√	√	x	√
Excluir um arquivo	√	√	x	√
Consultar uma lista de arquivos	√	√	√	√
Operar tags de cluster em lotes	√	√	x	√
Criar uma única tag de cluster	√	√	x	√
Excluir uma única tag de cluster	√	√	x	√
Consultar uma lista de recursos por tag	√	√	√	√
Consultar tags de cluster	√	√	√	√
Acessar o Manager	√	√	x	√
Consultar uma lista de patches	√	√	√	√
Instalar um patch	√	√	x	√
Desinstalar um patch	√	√	x	√
Autorizar canais de O&M	√	√	x	√

Operação	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
Compartilhar logs de canal de O&M	√	√	x	√
Consultar uma lista de alarmes	√	√	√	√
Subscrever a notificação de alarme	√	√	x	√
Submeter uma instrução SQL	√	√	x	√
Consultar resultados de SQL	√	√	x	√
Cancelar uma tarefa de execução 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"
        }
    ]
}
```

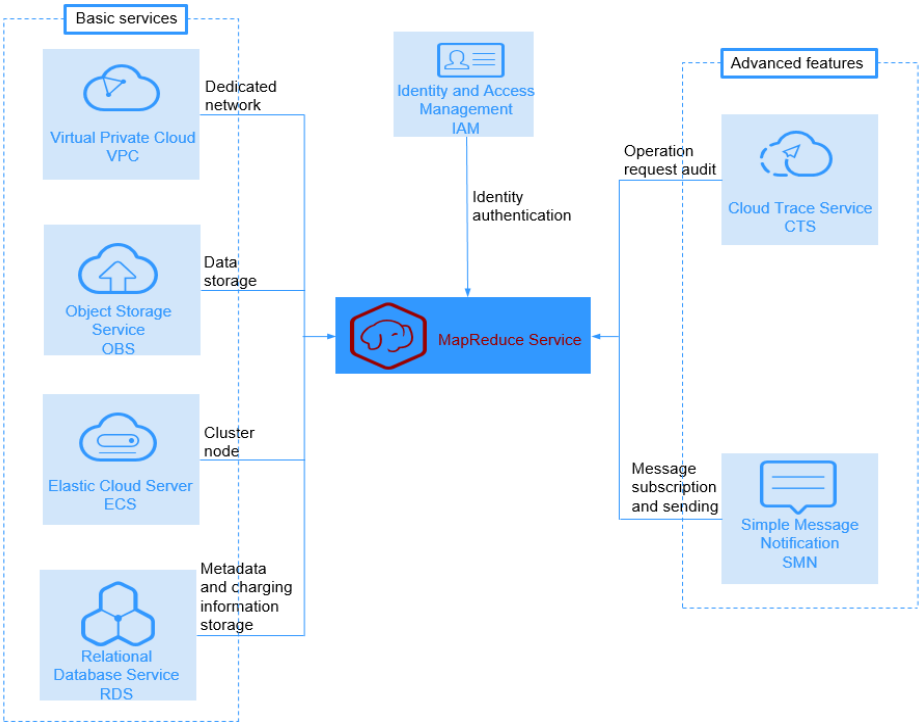
Links úteis

- **Visão geral do serviço IAM**
- **Criação de grupos de usuários e concessão de permissões de MRS**
- **Políticas de permissões e ações suportadas**

12 Serviços relacionados

Figura 12-1 mostra a relação entre o MRS e outros serviços.

Figura 12-1 Relações com outros serviços



Relações com outros serviços

Tabela 12-1 Relações com outros serviços

Serviço	Relações	Referência
Virtual Private Cloud (VPC)	Os clusters do MRS são criados nas sub-redes de uma VPC. As VPCs fornecem um ambiente de rede seguro, isolado e lógico para seus clusters do MRS.	**Criação de uma VPC e sub-rede**

Serviço	Relações	Referência
Object Storage Service (OBS)	O OBS armazena os seguintes dados do usuário: ● Dados de entrada de job do MRS, como programas do usuário e arquivos de dados ● Dados de saída de job do MRS, como arquivos de resultados e arquivos de log de jobs Em clusters do MRS, HDFS, Hive, MapReduce, YARN, Spark, Flume e Loader podem importar ou exportar dados do OBS. O MRS usa o sistema de arquivos paralelo do OBS para fornecer serviços.	**Configuração de um cluster desacoplado de armazenamento-computação (Agência)** **Configuração de um cluster desacoplado de armazenamento-computação (AK/SK)**
Elastic Cloud Server (ECS)	O MRS usa servidores de nuvem elásticos (ECSs) como nós de cluster.	**Preparação de um ambiente operacional** **Criação de um cluster**
Relational Database Service (RDS)	O RDS armazena dados em execução do sistema MRS, incluindo metadados de cluster do MRS e informações de faturamento do usuário.	**Configuração de conexões de dados**
Identity and Access Management (IAM)	O IAM fornece autenticação para MRS.	**Criação de um usuário e concessão de permissões** **Criação de políticas personalizadas do MRS** **Sincronização de usuários IAM para MRS**
Simple Message Notification (SMN)	MRS usa SMN para fornecer um-para-múltiplos mensagem subscrição e notificação sobre uma variedade de protocolos.	**Configuração de regras de notificação de job**
Cloud Trace Service (CTS)	O CTS fornece registros de operação de solicitações de operação de recursos do MRS e resultados de solicitações para consulta, auditoria e rastreamento inverso.	**Tabela 12-2**

Tabela 12-2 Operações do MRS gravadas pelo CTS

Operação	Tipo de recurso	Nome do rastreamento
Criar um cluster	cluster_mrs	createCluster
Excluir um cluster	cluster_mrs	deleteCluster
Expandir um cluster	cluster_mrs	scaleOutCluster
Encolher um cluster	cluster_mrs	scaleInCluster

Depois que você habilita o CTS, o sistema começa a gravar operações em recursos da nuvem. Você pode exibir os registros de operação dos últimos 7 dias no console de gerenciamento do CTS. Para obter detalhes, consulte **Cloud Trace Service** > **Primeiros passos** > **Consulta de rastreamentos em tempo real**.

13 Descrição da cota

As cotas de recursos disponíveis são configuradas para cada conta de usuário em um ambiente para evitar abuso de recursos.

A seguir lista a infraestrutura usada pelo MapReduce. As cotas são gerenciadas por cada serviço básico. Se você precisar aumentar as cotas, entre em contato com o suporte técnico do serviço correspondente.

- ECS
- BMS
- VPC
- EVS
- Image Management Service (IMS)
- OBS
- EIP
- SMN
- IAM

Para obter detalhes sobre como exibir e modificar cotas, consulte **Cotas**.

14 Conceitos comuns

Tabela HBase

Uma tabela HBase é um mapa tridimensional composto por uma ou mais colunas ou linhas de dados.

Coluna

Coluna é uma dimensão de uma tabela HBase. O nome da coluna está no formato de <*family*>**:**<*label*>, onde <*family*> e <*label*> podem ser qualquer combinação de caracteres. Uma tabela HBase consiste em um conjunto de famílias de colunas. Cada coluna na tabela HBase pertence a uma família de colunas.

Família de colunas

Uma família de colunas é uma coleção de colunas armazenadas no esquema HBase. Para criar colunas, primeiro você deve criar uma família de colunas. Uma família de colunas organiza dados com a mesma propriedade no HBase. Cada linha de dados na mesma família de colunas é armazenada no mesmo servidor. Cada família de colunas pode ser um atributo, como pacotes compactados, carimbos de data/hora e cache de blocos de dados.

MemStore

O MemStore é o núcleo do armazenamento de HBase. Quando a quantidade de dados armazenados no WAL atinge o limite superior, os dados são carregados em MemStore para classificação e armazenamento.

RegionServer

RegionServer é um serviço em execução em cada DataNode no cluster de HBase. Ele é responsável por atender e gerenciar regiões, fazer upload das informações de carga das regiões e gerenciar nós mestres distribuídos.

Carimbo de data/hora

Um carimbo de data/hora é um número inteiro de 64-bit usado para indexar diferentes versões dos mesmos dados. Um carimbo de data/hora pode ser atribuído automaticamente pelo HBase quando os dados são gravados ou atribuídos pelos usuários.

Store

Store é um núcleo do armazenamento HBase. Um Store hospeda um MemStore e vários StoreFiles. Um Store corresponde a uma família de colunas de uma tabela em uma região.

Índice

Um índice é uma estrutura de dados que melhora a eficiência da recuperação de dados em uma tabela de banco de dados. Uma ou mais colunas em uma tabela de banco de dados podem ser usadas para recuperação aleatória rápida de dados e acesso eficiente a registros ordenados.

Coprocessador

Um coprocessador é uma interface fornecida pelo HBase para implementar a lógica de cálculo no RegionServer. Os coprocessadores são classificados em coprocessadores de sistema e coprocessadores de tabela. O primeiro pode importar todas as tabelas de dados no RegionServer e o segundo pode processar uma tabela especificada.

Pool de blocos

Um pool de blocos é uma coleção de blocos que pertencem a um único namespace. DataNodes armazena blocos de todos os pools de blocos em um cluster. Cada pool de blocos é gerenciado de forma independente, o que permite que um namespace gere um ID para um novo bloco sem depender de outros namespaces. Se um NameNode for inválido, o DataNode ainda poderá fornecer serviços para outros NameNodes no cluster.

DataNode

Um DataNode é um nó de trabalho no cluster de HDFS. Programados pelo cliente ou NameNode, os DataNodes armazenam e recuperam dados e periodicamente relatam blocos de arquivos para NameNodes.

Bloco de arquivo

Um bloco de arquivo é a unidade lógica mínima armazenada no HDFS. Cada arquivo de HDFS é armazenado em um ou mais blocos de arquivos. Todos os blocos de arquivos são armazenados em DataNodes.

Réplica de bloco

Uma réplica é uma cópia em bloco armazenada no HDFS. Um bloco de arquivos armazena várias réplicas para a disponibilidade do sistema e a tolerância a falhas.

Volume de namespace

Um volume de namespace é uma unidade de gerenciamento independente que consiste em um namespace e seu pool de blocos. Quando um NameNode é excluído, os pools de blocos relacionados no DataNode também são excluídos. Durante uma atualização de cluster, cada volume de namespace é atualizado como um todo.

NodeManager

O NodeManager executa aplicações, monitora o uso de recursos (incluindo CPUs, memória, discos e recursos de rede) de aplicações e relata o uso de recursos ao ResourceManager.

ResourceManager

O ResourceManager agenda os recursos exigidos pelas aplicações. Ele fornece um plug-in de agendamento para alocação de recursos de cluster para várias filas e aplicações. O plug-in de agendamento agenda recursos com base em recursos existentes ou usando o modelo de agendamento justo.

Partição

Cada tópico pode ser dividido em várias partições. Cada partição corresponde a um arquivo de log anexado cuja sequência é fixa.

Seguidor

Um seguidor processa solicitações de leitura e trabalha com um líder para processar solicitações de gravação. Ele também pode ser usado como um líder de backup. Quando o líder é defeituoso, um seguidor é eleito para assumir a carga de trabalho do líder para evitar um único ponto de falha.

Observador

Os observadores não participam na votação para a eleição e escrevem solicitações. Eles só processam solicitações de leitura e encaminham solicitações de gravação para o líder, melhorando a eficiência do processamento.

Líder

Um líder dos clusters de ZooKeeper é eleito pelos seguidores usando o protocolo Zookeeper Atomic Broadcast (ZAB). Ele recebe e agenda todas as solicitações de gravação e sincroniza informações escritas para seguidores e observadores.

CarbonData

Carbon é uma arquitetura aberta baseada em Spark SQL. Ele integra o mecanismo MOLAP desenvolvido pela Huawei e o Spark, e constrói rapidamente o mecanismo de análise multidimensional distribuído baseado em Spark, encurtando a duração da análise de minutos para segundos e fortalecendo a capacidade de análise multidimensional do Spark.

DStream

DStream é um conceito abstrato fornecido por Spark Streaming. É um fluxo de dados contínuo que é obtido a partir da fonte de dados ou do fluxo de entrada transformado. Em essência, um DStream é uma série de conjuntos de dados distribuídos resilientes contínuos (RDDs).

Memória heap

Um heap indica a área de dados onde a Máquina Virtual Java (JVM) está em execução e da qual a memória para todas as instâncias de classe e matrizes é confirmada. Os parâmetros de

inicialização da JVM **-Xms** e **-Xmx** são usados para definir a memória heap inicial e a memória heap máxima, respectivamente.

- Máxima memória heap: memória heap que pode ser comprometida com um programa no máximo pelo sistema, que é especificada pelo parâmetro **-Xmx**.
- Memória heap comprometida: memória heap total comprometida pelo sistema para executar um programa. Ela varia entre a memória heap inicial e a memória heap máxima.
- Memória heap usada: memória heap usada por um programa. É menor que a memória heap comprometida.
- Memória não-heap: memória excluída dos heaps da JVM e da área de memória para executar a JVM. A memória não-heap tem os seguintes três pools de memória:
  - Code Cache: armazena código compilado JIT. Seu valor é definido por meio do parâmetro de inicialização da JVM **-XX:InitialCodeCacheSize -XX:ReservedCodeCacheSize**. O valor padrão é 240 MB.
  - Espaço de classe comprimido: armazena metadados de um ponteiro. Seu valor é definido por meio do parâmetro de inicialização da JVM **-XX:CompressedClassSpaceSize**. O valor padrão é 1024 MB.
  - Metaespaço: armazena metadados. Seu valor é definido através do parâmetro de inicialização da JVM **-XX:MetaspaceSize -XX:MaxMetaspaceSize**.
- Máxima memória não heap: memória não heap comprometida com um programa no máximo pelo sistema. Seu valor é a soma dos valores máximos de Cache de código, Espaço de classe comprimido e Metaespaço.
- Memória não-heap comprometida: memória não-heap total comprometida pelo sistema para executar um programa. Ela varia da memória não-heap inicial e a memória não-heap máxima.
- Memória não-heap utilizada: memória não-heap utilizada por um programa. É menor do que a memória não heap comprometida.

Hadoop

O Hadoop é uma estrutura de sistema distribuído. Ele permite que os usuários desenvolvam aplicativos distribuídos usando computação e armazenamento de alta velocidade fornecidos por clusters sem conhecer os detalhes subjacentes do sistema distribuído. Ele também pode processar de forma confiável e eficiente grandes quantidades de dados em modo escalável e distribuído. O Hadoop é confiável porque mantém várias duplicatas de dados de trabalho, permitindo o processamento distribuído de nós com falha. O Hadoop é altamente eficiente porque processa dados em modo paralelo. O Hadoop é escalável porque pode processar petabytes de dados. O Hadoop é composto por HDFS, HBase, MapReduce e Hive.

Função

Uma função é um elemento de um serviço. Um serviço contém uma ou várias funções. Os serviços são instalados nos servidores por meio de funções para que possam ser executados corretamente.

Cluster

Um cluster é uma tecnologia de computador que permite que vários servidores funcionem como um servidor. Os clusters melhoram a estabilidade, a confiabilidade e a capacidade de processamento de dados ou serviço do sistema. Por exemplo, os clusters podem impedir ponto

único de falhas (SPOFs), compartilhar recursos de armazenamento, reduzir a carga do sistema e melhorar o desempenho do sistema.

Instância

Uma instância é formada quando uma função de serviço é instalada no host. Um serviço tem uma ou mais instâncias de função.

Metadados

Metadados são dados que fornecem informações sobre outros dados e também são chamados de dados de mídia ou dados de retransmissão. Eles são usados para definir propriedades de dados, especificar locais de armazenamento de dados e dados históricos, recuperar recursos e registrar arquivos.

