MapReduce 服务

产品介绍

文档版本 01

发布日期 2025-08-26

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 什么是 MapReduce 服务

大数据是人类进入互联网时代以来面临的一个巨大问题：社会生产生活产生的数据量越来越大，数据种类越来越多，数据产生的速度越来越快。传统的数据处理技术，比如说单机存储，关系数据库已经无法解决这些新的大数据问题。为解决以上大数据处理问题，Apache基金会推出了Hadoop大数据处理的开源解决方案。Hadoop是一个开源分布式计算平台，可以充分利用集群的计算和存储能力，完成海量数据的处理。企业自行部署Hadoop系统有成本高，周期长，难运维和不灵活等问题。

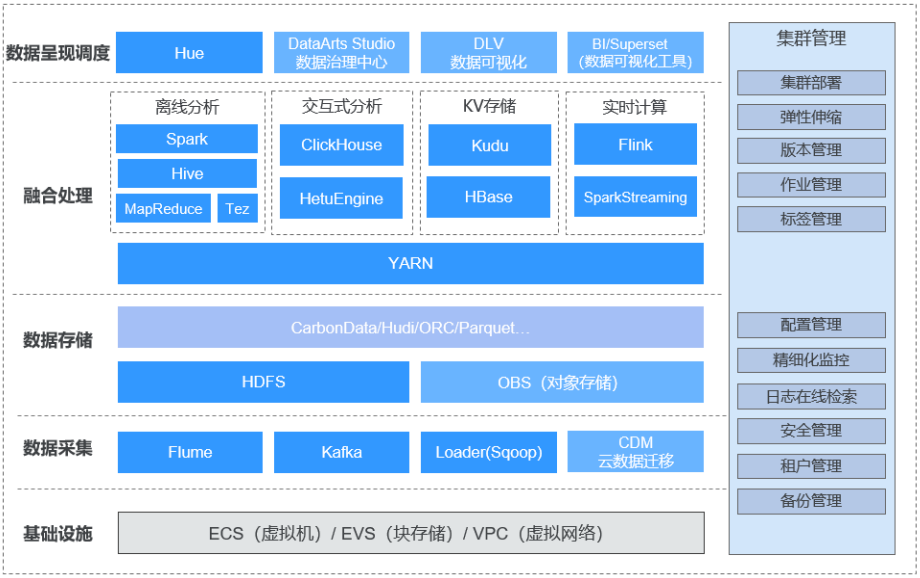
针对上述问题，华为云提供了大数据MapReduce服务（MRS），MRS是一个在华为云上部署和管理Hadoop系统的服务，一键即可部署Hadoop集群。MRS提供租户完全可控的一站式企业级大数据集群云服务，完全兼容开源接口，结合华为云计算、存储优势及大数据行业经验，为客户提供高性能、低成本、灵活易用的全栈大数据平台，轻松运行Hadoop、Spark、HBase、Kafka、Storm等大数据组件，并具备在后续根据业务需要进行定制开发的能力，帮助企业快速构建海量数据信息处理系统，并通过对海量信息数据实时与非实时的分析挖掘，发现全新价值点和企业商机。

产品架构

MRS集群各版本组件情况请参见**MRS组件版本一览表**。

MRS逻辑架构如**图1-1**所示。

图 1-1 MRS 架构



MRS架构包括了基础设施和大数据处理流程各个阶段的能力。

- 基础设施

  MRS基于华为云弹性云服务器ECS构建的大数据集群，充分利用了其虚拟化层的高可靠、高安全的能力。

  - 虚拟私有云（VPC）为每个租户提供的虚拟内部网络，默认与其他网络隔离。
  - 云硬盘（EVS）提供高可靠、高性能的存储。
  - 弹性云服务器（ECS）提供的弹性可扩展虚拟机，结合VPC、安全组、EVS数据多副本等能力打造一个高效、可靠、安全的计算环境。

- 数据采集

  数据采集层提供了数据接入到MRS集群的能力，包括Flume（数据采集）、Loader（关系型数据导入）、Kafka（高可靠消息队列），支持各种数据源导入数据到大数据集群中。使用云数据迁移云服务也可以将外部数据导入至MRS集群中。

- 数据存储

  MRS支持结构化和非结构化数据在集群中的存储，并且支持多种高效的格式来满足不同计算引擎的要求。

  - HDFS是大数据上通用的分布式文件系统。
  - OBS是对象存储服务，具有高可用低成本的特点。

- 数据融合处理

  - MRS提供多种主流计算引擎：MapReduce（批处理）、Tez（DAG模型）、Spark（内存计算）、SparkStreaming（微批流计算）、Storm（流计算）、Flink（流计算），满足多种大数据应用场景，将数据进行结构和逻辑的转换，转化成满足业务目标的数据模型。
  - 基于预设的数据模型，使用易用的SQL数据分析，用户可以选择Hive（数据仓库），SparkSQL以及Presto交互式查询引擎。

- 数据呈现调度

用于数据分析结果的呈现，并与数据治理中心DataArts Studio集成，提供一站式的大数据协同开发平台，帮助用户轻松完成数据建模、数据集成、脚本开发、作业调度、运维监控等多项任务，可以极大降低用户使用大数据的门槛，帮助用户快速构建大数据处理中心。

- 集群管理

  以Hadoop为基础的大数据生态的各种组件均是以分布式的方式进行部署，其部署、管理和运维复杂度较高。

  MRS集群管理提供了统一的运维管理平台，包括一键式部署集群能力，并提供多版本选择，支持运行过程中集群在无业务中断条件下，进行扩缩容、弹性伸缩。同时MRS集群管理还提供了作业管理、资源标签管理，以及对上述数据处理各层组件的运维，并提供监控、告警、参数配置、补丁升级等一站式运维能力。

产品优势

MRS服务拥有强大的Hadoop内核团队，基于华为FusionInsight大数据企业级平台构筑。历经行业数万节点部署量的考验，提供多级用户SLA保障。

MRS具有如下优势：

- 高性能

  MRS支持自研的CarbonData存储技术。CarbonData是一种高性能大数据存储方案，以一份数据同时支持多种应用场景，并通过多级索引、字典编码、预聚合、动态Partition、准实时数据查询等特性提升了IO扫描和计算性能，实现万亿数据分析秒级响应。同时MRS支持自研增强型调度器Superior，突破单集群规模瓶颈，单集群调度能力超10000节点。

- 低成本

  基于多样化的云基础设施，提供了丰富的计算、存储设施的选择，同时计算存储分离，提供了低成本海量数据存储方案。MRS可以按业务峰谷，自动弹性伸缩，帮助客户节省大数据平台闲时资源。MRS集群可以用时再创建、用时再扩容，用完就可以删除、缩容，确保低成本。

- 高安全

  MRS服务拥有企业级的大数据多租户权限管理能力，拥有企业级的大数据安全管理特性，支持按照表/按列控制访问权限，支持数据按照表/按列加密。

- 易运维

  MRS提供可视化大数据集群管理平台，提高运维效率。并支持滚动补丁升级，可视化补丁发布信息，一键式补丁安装，无需人工干预，不停业务，保障用户集群长期稳定。

- 高可靠

  MRS服务经过大规模的可靠性、长稳验证，满足企业级高可靠要求，同时支持数据跨AZ/跨Region自动备份的数据容灾能力，自动反亲和技术，虚拟机分布在不同物理机上。

首次使用 MRS

如果您是首次使用MRS的用户，建议您学习并了解如下信息：

- **基础知识了解**

  通过MRS**组件介绍**和**产品功能**章节的内容，了解MRS相关的基础知识，包含MRS各组件的基本原理和增强特性介绍，以及MRS服务的特有概念和功能的详细介绍。

- **入门使用**

  您可以参考**《快速入门》**学习并上手使用MRS。《快速入门》提供了样例的详细操作指导，您可以基于此操作指导，创建和使用MRS集群。

- **使用更多的功能，并查看其相关操作指导**

  如果您是一个MRS集群使用和运维人员，可以参考**用户指南**完成集群的生命周期管理、扩缩容以及作业管理等操作。集群中组件的使用指导可以详细参考**组件操作指南**。

  如果您是一个开发者，可以参考MRS提供的**开发指南**操作指导及样例工程开发并运行调测自己的应用程序。您也可以通过API调用完成MRS集群管理、作业执行等相关操作，您可以参考**《API参考》**获取详情。

2 产品优势

MapReduce服务（MRS）提供租户完全可控的企业级大数据集群云服务，轻松运行Hadoop、Spark、HBase、Kafka、ClickHouse等大数据组件，用户无需关注硬件的购买和维护。MRS服务拥有强大的Hadoop内核团队，基于华为大数据企业级平台构筑，历经行业数万节点部署量的考验，提供多级用户SLA保障。与自建Hadoop集群相比，MRS还具有以下优势：

1. **MRS支持一键式创建、删除、扩缩容集群，并通过弹性公网IP便捷访问MRS集群管理系统，让大数据集群更加易于使用**。
   - 用户自建大数据集群面临成本高、周期长、运维难和不灵活等问题。针对这些问题，MRS支持一键式创建、删除、扩容和缩容集群的能力，用户可以自定义集群的类型，组件范围，各类型的节点数、虚拟机规格、可用区、VPC网络、认证信息，MRS将为用户自动创建一个符合配置的集群，全程无需用户参与。同时支持用户快速创建多应用场景集群，比如快速创建Hadoop分析集群、HBase集群、Kafka集群。MRS支持部署异构集群，在集群中存在不同规格的虚拟机，允许在CPU类型，硬盘容量，硬盘类型，内存大小灵活组合。
   - MRS提供了基于弹性公网IP来便捷访问组件WebUI的安全通道，并且比用户自己绑定弹性公网IP更便捷，只需界面鼠标操作，即可简化原先用户需要自己登录虚拟私有云添加安全组规则，获取公网IP等步骤，减少了用户操作步骤。
   - MRS提供了自定义引导操作，用户可以以此为入口灵活配置自己的集群，通过引导操作用户可以自动化地完成安装MRS还没支持的第三方软件，修改集群运行环境等自定义操作。
   - MRS支持WrapperFS特性，提供OBS的翻译能力，兼容HDFS到OBS的平滑迁移，解决用户将HDFS中的数据迁移到OBS后，即可实现客户端无需修改自己的业务代码逻辑的情况下，访问存储到OBS的数据。
2. **MRS支持自动弹性伸缩，相对自建Hadoop集群的使用成本更低**。

   MRS可以按业务峰谷，自动弹性伸缩，在业务繁忙时申请额外资源，业务不繁忙时释放闲置资源，让用户按需使用，帮助用户节省大数据平台闲时资源，尽可能地帮助用户降低使用成本，聚焦核心业务。

   在大数据应用，尤其是周期性的数据分析处理场景中，需要根据业务数据的周期变化，动态调整集群计算资源以满足业务需要。MRS的弹性伸缩规则功能支持根据集群负载对集群进行弹性伸缩。此外，如果数据量为周期有规律的变化，并且希望在数据量变化前提前完成集群的扩缩容，可以使用MRS的资源计划特性。MRS服务支持规则和时间计划两种弹性伸缩的策略：

- 弹性伸缩规则：根据集群实时负载对Task节点数量进行调整，数据量变化后触发扩缩容，有一定的延后性。
- 资源计划：若数据量变化存在周期性规律，则可通过资源计划在数据量变化前提前完成集群的扩缩容，避免出现增加或减少资源的延后。

弹性伸缩规则与资源计划均可触发弹性伸缩，两者既可同时配置也可单独配置。资源计划与基于负载的弹性伸缩规则叠加使用可以使得集群节点的弹性更好，足以应对偶尔超出预期的数据峰值出现。

3. **MRS支持存算分离，大幅提升大数据集群资源利用率**。

   针对传统存算一体大数据架构中扩容困难、资源利用率低等问题，MRS采用计算存储分离架构，存储基于公有云对象存储实现11个9的高可靠，无限容量，支撑企业数据量持续增长；计算资源支持0~N弹性扩缩，百节点快速发放。存算分离后，计算节点可实现真正的极致弹性伸缩；数据存储部分基于OBS的跨AZ等能力实现更高可靠性，无需担心地震、挖断光纤等突发事件。存储和计算资源可以灵活配置，根据业务需要各自独立进行弹性扩展，可使资源匹配更精准、更合理，让大数据集群资源利用率大幅提升，综合分析成本降低50%。

   同时通过高性能的计算存储分离架构，打破存算一体架构并行计算的限制，最大化发挥对象存储的高带宽、高并发的特点，对数据访问效率和并行计算深度优化（元数据操作、写入算法优化等），实现性能提升。

4. **MRS支持自研的超级调度器Superior Scheduler，性能更优**。

   MRS支持自研超级调度器Superior Scheduler，突破单集群规模瓶颈，单集群调度能力超10000节点。Superior Scheduler是一个专门为Hadoop YARN分布式资源管理系统设计的调度引擎，是针对企业客户融合资源池，多租户的业务诉求而设计的高性能企业级调度器。Superior Scheduler可实现开源调度器、Fair Scheduler以及Capacity Scheduler的所有功能。另外，相较于开源调度器，Superior Scheduler在企业级多租户调度策略、租户内多用户资源隔离和共享、调度性能、系统资源利用率和支持大集群扩展性方面都做了针对性的增强，让Superior Scheduler直接替代开源调度器。

5. **MRS基于鲲鹏处理器进行软硬件垂直优化，充分释放硬件算力，实现高性价比**。

   MRS支持华为自研鲲鹏服务器，充分利用鲲鹏多核高并发能力，提供芯片级的全栈自主优化能力，使用华为自研的操作系统EulerOS、华为JDK及数据加速层，充分释放硬件算力，为大数据计算提供高算力输出。在性能相当情况下，端到端的大数据解决方案成本下降30%。

6. **MRS支持多种隔离模式及企业级的大数据多租户权限管理能力，安全性更高**。

   - MRS服务支持资源专属区内部署，专属区内物理资源隔离，用户可以在专属区内灵活地组合计算存储资源，包括专属计算资源+共享存储资源、共享计算资源+专属存储资源、专属计算资源+专属存储资源。MRS集群内支持逻辑多租户，通过权限隔离，对集群的计算、存储、表格等资源按租户划分。
   - MRS支持Kerberos安全认证，实现了基于角色的安全控制及完善的审计功能。
   - MRS支持对接华为云云审计服务（CTS），为用户提供MRS资源操作请求及请求结果的操作记录，供用户查询、审计和回溯使用。支持所有集群操作审计，所有用户行为可溯源。
   - MRS支持与主机安全服务对接，针对主机安全服务，做过兼容性测试，保证功能和性能不受影响的情况下，增强服务的安全能力。
   - MRS支持基于WebUI的统一的用户登录能力，Manager自带用户认证环节，用户只有通过Manager认证才能正常访问集群。
   - MRS支持数据存储加密，所有用户账号密码加密存储，数据通道加密传输，服务模块跨信任区的数据访问支持双向证书认证等能力。

- MRS大数据集群提供了完整的企业级大数据多租户解决方案。多租户是MRS大数据集群中的多个资源集合（每个资源集合是一个租户），具有分配和调度资源（资源包括计算资源和存储资源）的能力。多租户将大数据集群的资源隔离成一个个资源集合，彼此互不干扰，用户通过“租用”需要的资源集合，来运行应用和作业，并存放数据。在大数据集群上可以存在多个资源集合来支持多个用户的不同需求。
- MRS支持细粒度权限管理，结合华为云IAM服务提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。例如：针对MRS服务，管理员能够控制IAM用户仅能对集群进行指定的管理操作。如不允许某用户组删除集群，仅允许操作MRS集群基本操作，如创建集群、查询集群列表等。同时MRS支持多租户对OBS存储的细粒度权限管理，根据多种用户角色来区分访问OBS桶及其内部的对象的权限，实现MRS用户对OBS桶下的目录权限控制。
- MRS支持企业项目管理。企业项目是一种云资源管理方式，企业管理（Enterprise Management）提供面向企业客户的云上资源管理、人员管理、权限管理、财务管理等综合管理服务。区别于管理控制台独立操控、配置云产品的方式，企业管理控制台以面向企业资源管理为出发点，帮助企业以公司、部门、项目等分级管理方式实现企业云上的人员、资源、权限、财务的管理。MRS支持已开通企业项目服务的用户在创建集群时为集群配置对应的项目，然后使用企业项目管理对MRS上的资源进行分组管理。此特性适用于用户针对多个资源进行分组管理，并对相应的企业项目进行诸如权限控制、分项目费用查看等操作的场景。

7. **MRS管理节点均实现HA，支持完备的可靠性机制，让系统更加可靠**。

   MRS在基于Apache Hadoop开源软件的基础上，在主要业务部件的可靠性方面进行了优化和提升。

   - 管理节点均实现HA

     Hadoop开源版本的数据、计算节点已经是按照分布式系统进行设计的，单节点故障不影响系统整体运行；而以集中模式运作的管理节点可能出现的单点故障，就成为整个系统可靠性的短板。

     MRS对所有业务组件的管理节点都提供了类似的双机的机制，包括Manager、Presto、HDFS NameNode、Hive Server、HBase HMaster、YARN Resources Manager、Kerberos Server、Ldap Server等，全部采用主备或负荷分担配置，有效避免了单点故障场景对系统可靠性的影响。

   - 完备的可靠性机制

     通过可靠性分析方法，梳理软件、硬件异常场景下的处理措施，提升系统的可靠性。

     - 保障意外掉电时的数据可靠性，不论是单节点意外掉电，还是整个集群意外断电，恢复供电后系统能够正常恢复业务，除非硬盘介质损坏，否则关键数据不会丢失。
     - 硬盘亚健康检测和故障处理，对业务不造成实际影响。
     - 自动处理文件系统的故障，自动恢复受影响的业务。
     - 自动处理进程和节点的故障，自动恢复受影响的业务。
     - 自动处理网络故障，自动恢复受影响的业务。

8. **MRS提供统一的可视化大数据集群管理界面，让运维人员更加轻松**。

- MRS提供统一的可视化大数据集群管理界面，包括服务启停、配置修改、健康检查等能力，并提供可视化、便捷的集群管理监控告警功能；支持一键式系统运行健康度巡检和审计，保障系统的正常运行，降低系统运维成本。
- MRS联合消息通知服务(SMN)，在配置消息通知后，可以实时给用户发送MRS集群健康状态，用户可以通过手机短信或邮箱实时接收到MRS集群变更及组件告警信息，帮助用户轻松运维，实时监控，实时发送告警。
- MRS支持滚动补丁升级，可视化补丁发布信息，一键式补丁安装，无需人工干预，不停业务，保障用户集群长期稳定。
- MRS服务支持运维授权的功能，用户在使用MRS集群过程中，发生问题可以在MRS页面发起运维授权，由运维人员帮助用户快速定位问题，用户可以随时收回该授权。同时用户也可以在MRS页面发起日志共享，选择日志范围共享给运维人员，以便运维人员在不接触集群的情况下帮助定位问题。
- MRS支持将创建集群失败的日志转储到OBS，便于运维人员获取日志进行分析。

9. **MRS具有开放的生态，支持无缝对接周边服务，快速构建统一大数据平台。**
   - 以全栈大数据MRS服务为基础，企业可以一键式构筑数据接入、数据存储、数据分析和价值挖掘的统一大数据平台，并且与数据治理中心DataArts Studio及数据可视化等服务对接，为用户轻松解决数据通道上云、大数据作业开发调度和数据展现的困难，使用户从复杂的大数据平台构建和专业大数据调优和维护中解脱出来，更加专注行业应用，使用户完成一份数据多业务场景使用的诉求。DataArts Studio是数据全生命周期一站式开发运营平台，提供数据集成、数据开发、数据治理、数据服务、数据可视化等功能。MRS数据支持连接DataArts Studio平台，并基于可视化的图形开发界面、丰富的数据开发类型（脚本开发和作业开发）、全托管的作业调度和运维监控能力，内置行业数据处理pipeline，一键式开发，全流程可视化，支持多人在线协同开发，极大地降低了用户使用大数据的门槛，帮助用户快速构建大数据处理中心，对数据进行治理及开发调度，快速实现数据变现。
   - MRS服务100%兼容开源大数据生态，结合周边丰富的数据及应用迁移工具，能够帮助用户快速完成自建平台的平滑迁移，整个迁移过程可做到“代码0修改，业务0中断”。

3 应用场景

大数据在人们的生活中无处不在，在IoT、电子商务、金融、制造、医疗、能源和政府部门等行业均可以使用华为云MRS服务进行大数据处理。

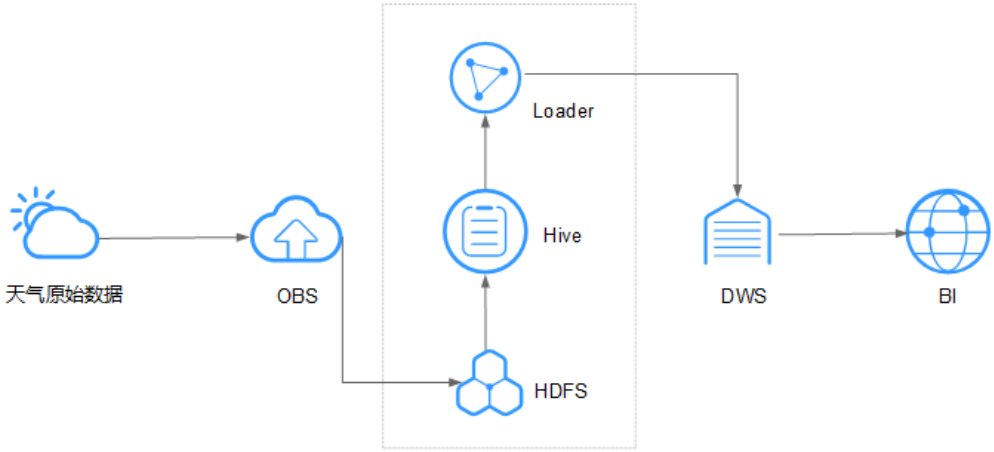
海量数据分析场景

海量数据分析是现代大数据系统中的主要场景。通常企业会包含多种数据源，接入后需要对数据进行ETL（Extract-Transform-Load）处理形成模型化数据，以便提供给各个业务模块进行分析梳理，这类业务通常有以下特点：

- 对执行实时性要求不高，作业执行时间在数十分钟到小时级别。
- 数据量巨大。
- 数据来源和格式多种多样。
- 数据处理通常由多个任务构成，对资源需要进行详细规划。

例如在环保行业中，可以将天气数据存储在OBS，定期转储到HDFS中进行批量分析，在1小时内MRS可以完成10TB的天气数据分析。

图 3-1 环保行业海量数据分析场景



该场景下MRS的优势如下所示。

- 低成本：利用OBS实现低成本存储。

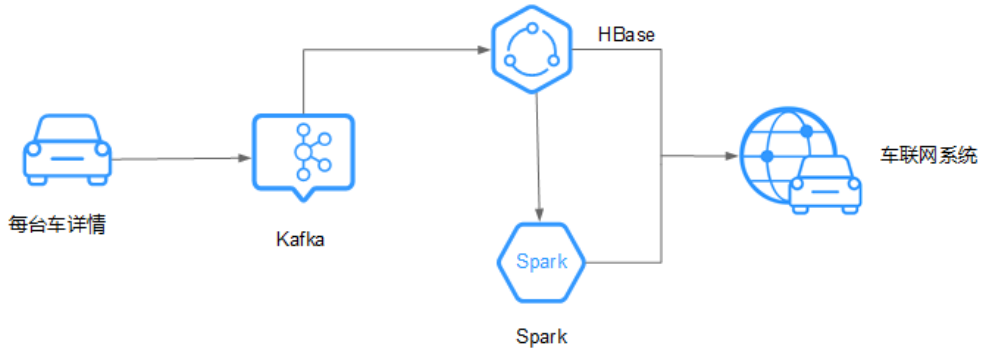
- 海量数据分析：利用Hive实现TB/PB级的数据分析。
- 可视化的导入导出工具：通过可视化导入导出工具Loader，将数据导出到DWS，完成BI分析。

海量数据存储场景

用户拥有大量结构化数据后，通常需要提供基于索引的准实时查询能力，如车联网场景下，根据汽车编号查询汽车维护信息，存储时，汽车信息会基于汽车编号进行索引，以实现该场景下的秒级响应。通常这类数据量比较庞大，用户可能保存1至3年的数据。

例如在车联网行业，某车企将数据储存在HBase中，以支持PB级别的数据存储和毫秒级的数据详单查询。

图 3-2 车联网行业海量数据存储场景



该场景下MRS的优势如下所示。

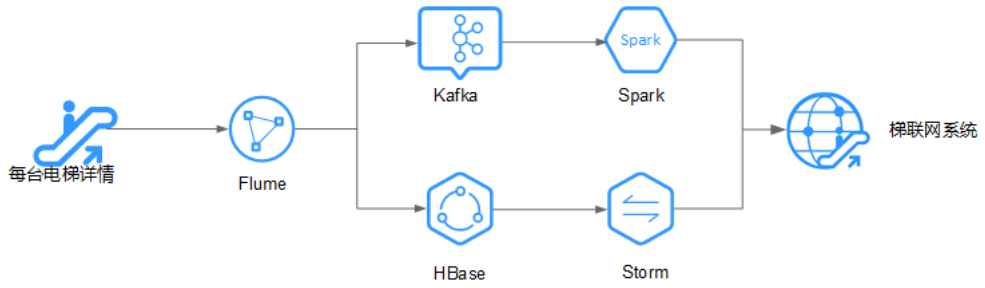
- 实时：利用Kafka实现海量汽车的消息实时接入。
- 海量数据存储：利用HBase实现海量数据存储，并实现毫秒级数据查询。
- 分布式数据查询：利用Spark实现海量数据的分析查询。

实时数据处理

实时数据处理通常用于异常检测、欺诈识别、基于规则告警、业务流程监控等场景，在数据输入系统的过程中，对数据进行处理。

例如在梯联网行业，智能电梯的数据，实时传入到MRS的流式集群中进行实时告警。

图 3-3 梯联网行业低时延流式处理场景



该场景下MRS的优势如下所示。

- 实时数据采集：利用Flume实现实时数据采集，并提供丰富的采集和存储连接方式。
- 海量的数据源接入：利用Kafka实现万级别的电梯数据的实时接入。

4 MRS 集群版本说明

MRS 集群版本类型

MRS集群版本类型分为普通版与LTS版本，不同版本集群所包含的组件内容及特性略有不同，用户可根据自身业务需求进行选择。

- **普通版**
  - **功能说明**

    普通版支持集群基础操作如配置、管理和运维等，具体可以查看**用户指南**。
  - **组件介绍**

    除共有组件外，普通版集群还支持Presto、Impala、Kudu、Sqoop等组件，可以根据不同集群版本选择不同的组件，具体各版本集群的组件详情可以参考**MRS组件版本一览表**和**组件操作指南**。
- **LTS版**
  - **功能说明**

    LTS版集群除支持集群基础操作外，还提供版本升级能力。如需使用该功能请联系技术支持。
  - **组件介绍**

    除共有组件外，LTS版集群还支持HetuEngine、IoTDB等组件，可以根据不同集群版本选择不同的组件，具体各版本集群的组件详情可以参考**MRS组件版本一览表**和**组件操作指南**。

MRS 集群版本选择建议

- LTS版集群支持版本升级能力，如果您需要使用版本升级能力，您可以选择购买LTS版集群。
- LTS版集群具备多可用区部署能力，可以实现集群可用区级别的容灾。如果您需要MRS集群具备更高的安全性能和容灾能力，您可以选择购买LTS版集群。
- LTS版集群支持HetuEngine等组件，如果您需要使用相关组件，您可以选择购买LTS版集群。

  说明

  由于已购买的LTS版集群无法切换为普通版，请根据需要选择购买。目前MRS普通版集群不支持直接购买，如需使用普通版集群请提交工单申请开通。

不同版本计费差异

普通版和LTS版由于功能不一致，计费存在一定差异，详情请查看**计费说明**，您也可以通过MRS提供的**价格计算器**，选择您需要的集群版本、节点规格，快速计算出购买MRS集群的参考价格。

5 MRS 组件版本一览表

组件及版本号信息

MRS各集群版本配套的组件及版本号信息如表5-1所示。

说明

- Hadoop组件包含HDFS、Yarn、Mapreduce服务，DBService、KrbServer及LdapServer等集群内部使用的组件，在创建集群时的组件列表中不呈现。
- MRS组件的版本号通常与组件开源版本号保持一致。
- MRS集群内各组件不支持单独升级，请根据实际需要选择对应版本的集群。
- LTS（Long Term Support）版本集群与普通版本集群区别可参考**MRS集群版本说明**。
- 部分集群版本为受限使用阶段，如需使用请提交工单申请开通。

表 5-1 MRS 组件版本信息

MRS支持的组件	MRS 1.9.x	MRS 3.1.0	MRS 3.1.2-LTS.x	MRS 3.1.5	MRS 3.2.0-LTS.x	MRS 3.3.0-LTS.x	MRS 3.3.1-LTS.x	MRS 3.5.0-LTS.x
Alluxio	2.0.1	-	-	-	-	-	-	-
ClickHouse	-	21.3.4.25	21.3.4.25	21.3.4.25	22.3.2.2	23.3.2.37	23.3.2.37	23.3.2.37
Doris	-	-	-	-	-	1.2.3	2.0.5	2.0.13
DBService	1.0.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0
Flink	1.7.0	1.12.0	1.12.2	1.12.2	1.15.0	1.15.0	1.17.1	1.17.1
Flume	1.6.0	1.9.0	1.9.0	1.9.0	1.9.0	1.11.0	1.11.0	1.11.0
Guardian	-	-	-	0.1.0	-	0.1.0	0.1.0	0.1.0
HBase	1.3.1	2.2.3	2.2.3	2.2.3	2.2.3	2.4.14	2.4.14	2.4.14

MRS支持的组件	MRS 1.9.x	MRS 3.1.0	MRS 3.1.2-LTS.x	MRS 3.1.5	MRS 3.2.0-LTS.x	MRS 3.3.0-LTS.x	MRS 3.3.1-LTS.x	MRS 3.5.0-LTS.x
HDFS	2.8.3	3.1.1	3.1.1	3.1.1	3.3.1	3.3.1	3.3.1	3.3.1
HetuEngine	-	-	1.2.0	-	1.2.0	2.0.0	2.0.0	2.1.0
Hive	2.3.3	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0	3.1.0
Hudi（集成在Spark中）	-	0.7.0	0.9.0	0.9.0	0.11.0	0.11.0	0.11.0	0.15.0
Hue	3.11.0	4.7.0	4.7.0	4.7.0	4.7.0	-	-	-
Impala	-	3.4.0	-	3.4.0	-	-	4.3.0	4.3.0
IoTDB	-	-	-	-	0.14.0	-	-	-
Kafka	1.1.0	2.11-2.4.0	2.11-2.4.0	2.11-2.4.0	2.11-2.4.0	2.12-2.8.1	2.12-3.6.1	2.12-3.6.1
KafkaManager	1.3.3.1	-	-	-	-	-	-	-
KrbServer	1.15.2	1.17	1.18	1.18	1.18	1.20	1.20	1.20
Kudu	-	1.12.1	-	1.12.1	-	-	1.17.0	1.17.0
LdapServer	1.0.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0	2.7.0
Loader	2.0.0	-	1.99.3	-	1.99.3	1.99.3	-	-
MapReduce	2.8.3	3.1.1	3.1.1	3.1.1	3.3.1	3.3.1	3.3.1	3.3.1
Oozie	-	5.1.0	5.1.0	5.1.0	5.1.0	5.1.0	5.1.0	5.1.0
Opentsdb	2.3.0	-	-	-	-	-	-	-
Presto	0.216	333	-	333	-	-	-	-
Phoenix（集成在HBase中）	-	5.0.0	5.0.0	5.0.0	5.0.0	5.1.2	5.1.2	5.1.2

MRS支持的组件	MRS 1.9.x	MRS 3.1.0	MRS 3.1.2-LTS.x	MRS 3.1.5	MRS 3.2.0-LTS.x	MRS 3.3.0-LTS.x	MRS 3.3.1-LTS.x	MRS 3.5.0-LTS.x
Ranger	1.0.1	2.0.0	2.0.0	2.0.0	2.0.0	2.3.0	2.3.0	2.3.0
Spark/Spark2x	2.2.2	2.4.5	3.1.1	3.1.1	3.1.1	3.3.1	3.3.1	3.3.1
Sqoop	-	1.4.7	-	1.4.7	-	-	1.4.7	1.4.7
Storm	1.2.1	-	-	-	-	-	-	-
Tez	0.9.1	0.9.2	0.9.2	0.9.2	0.9.2	0.10.2	0.10.2	0.10.2
Yarn	2.8.3	3.1.1	3.1.1	3.1.1	3.3.1	3.3.1	3.3.1	3.3.1
ZooKeeper	3.5.1	3.5.6	3.6.3	3.6.3	3.6.3	3.8.1	3.8.1	3.8.1
MRS Manager	1.9.2	8.1.0	8.1.2.x	8.1.2	8.2.0.x	8.3.0.x	8.3.1.x	8.5.0.x

组件及版本号信息（已下线版本）

MRS已下线集群版本配套的组件及版本号信息如表5-2所示。

表 5-2 MRS 组件版本信息（已下线版本）

MRS支持的组件	MRS 3.0.5
Alluxio	2.3.0
CarbonData	2.0.1
ClickHouse	21.3.4.25
DBService	2.7.0
Flink	1.10.0
Flume	1.9.0
HBase	2.2.3
HDFS	3.1.1
Hive	3.1.0
Hue	4.7.0
Impala	3.4.0

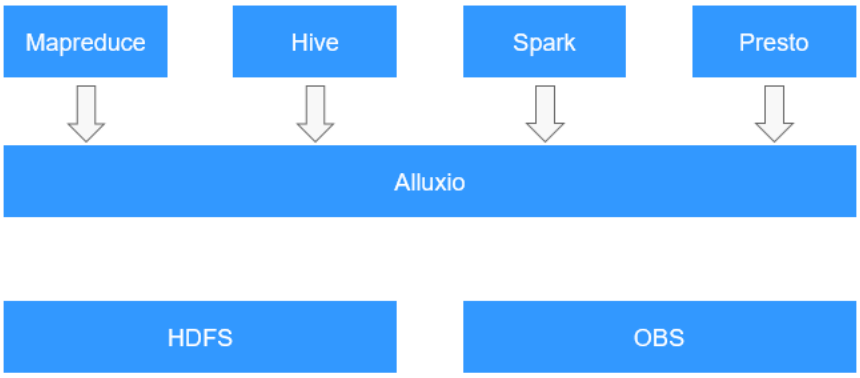
MRS支持的组件	MRS 3.0.5
Kafka	2.11-2.4.0
KafkaManager	-
KrbServer	1.17
Kudu	1.12.1
LdapServer	2.7.0
Loader	1.99.3
MapReduce	3.1.1
Oozie	5.1.0
Opentsdb	-
Presto	333
Phoenix	5.0.0
Ranger	2.0.0
Spark	-
Spark2x	2.4.5
Storm	1.2.1
Tez	0.9.2
YARN	3.1.1
ZooKeeper	3.5.6
MRS Manager	-
FusionInsight Manager	8.0.2.1

6 组件介绍

6.1 Alluxio

Alluxio是一个面向基于云的数据分析和人工智能的数据编排技术。在MRS的大数据生态系统中，Alluxio位于计算和存储之间，为包括Apache Spark、Presto、Mapreduce和Apache Hive的计算框架提供了数据抽象层，使上层的计算应用可以通过统一的客户端API和全局命名空间访问包括HDFS和OBS在内的持久化存储系统，从而实现了对计算和存储的分离。

图 6-1 Alluxio 架构



优势：

- 提供内存级I/O吞吐率，同时降低具有弹性扩张特性的数据驱动型应用的成本开销
- 简化云存储和对象存储接入
- 简化数据管理，提供对多数据源的单点访问
- 应用程序部署简易

有关Alluxio的详细信息，请参见：**https://docs.alluxio.io/os/user/stable/cn/Overview.html**。

6.2 ClickHouse

6.2.1 ClickHouse 基本原理

ClickHouse 简介

ClickHouse是一款开源的面向联机分析处理的列式数据库，其独立于Hadoop大数据体系，最核心的特点是压缩率和极速查询性能。同时，ClickHouse支持SQL查询，且查询性能好，特别是基于大宽表的聚合分析查询性能非常优异，比其他分析型数据库速度快一个数量级。

更多关于ClickHouse组件操作指导，请参考**使用ClickHouse**。

ClickHouse核心的功能特性介绍如下：

完备的DBMS功能

ClickHouse拥有完备的**DBMS**数据库管理系统（Database Management System），基本功能如下所示。

- DDL（数据定义语言）：可以动态地创建、修改或删除数据库、表和视图，而无须重启服务。
- DML（数据操作语言）：可以动态查询、插入、修改或删除数据。
- 权限控制：可以按照用户粒度设置数据库或者表的操作权限，保障数据的安全性。
- 数据备份与恢复：提供了数据备份导出与导入恢复机制，满足生产环境的要求。
- 分布式管理：提供集群模式，能够自动管理多个数据库节点。

列式存储与数据压缩

ClickHouse是一款使用列式存储的数据库，数据按列进行组织，属于同一列的数据会被保存在一起，列与列之间也会由不同的文件分别保存。

在执行数据查询时，列式存储可以减少数据扫描范围和数据传输时的大小，提高了数据查询的效率。

例如在传统的行式数据库系统中，数据按如下**表6-1**顺序存储：

表 6-1 行式数据库

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

行式数据库中处于同一行中的数据总是被物理的存储在一起，而在列式数据库系统中，数据按如下表6-2顺序存储：

表 6-2 列式数据库

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

该示例中只展示了数据在列式数据库中数据的排列方式。对于存储而言，列式数据库总是将同一列的数据存储在一起，不同列的数据也总是分开存储，列式数据库更适合于OLAP（Online Analytical Processing）场景。

向量化执行引擎

ClickHouse利用CPU的SIMD指令实现了向量化执行。SIMD的全称是Single Instruction Multiple Data，即用单条指令操作多条数据，通过数据并行以提高性能的一种实现方式（其他的还有指令级并行和线程级并行），它的原理是在CPU寄存器层面实现数据的并行操作。

关系模型与SQL查询

ClickHouse完全使用SQL作为查询语言，提供了标准协议的SQL查询接口，使得现有的第三方分析可视化系统可以轻松与它集成对接。

同时ClickHouse使用了关系模型，所以将构建在传统关系型数据库或数据仓库之上的系统迁移到ClickHouse的成本会变得更低。

数据分片与分布式查询

ClickHouse集群由1到多个分片组成，而每个分片则对应了ClickHouse的1个服务节点。分片的数量上限取决于节点数量（1个分片只能对应1个服务节点）。

ClickHouse提供了本地表（Local Table）与分布式表（Distributed Table）的概念。一张本地表等同于一份数据的分片。而分布式表本身不存储任何数据，它是本地表的访问代理，其作用类似分库中间件。借助分布式表，能够代理访问多个数据分片，从而实现分布式查询。

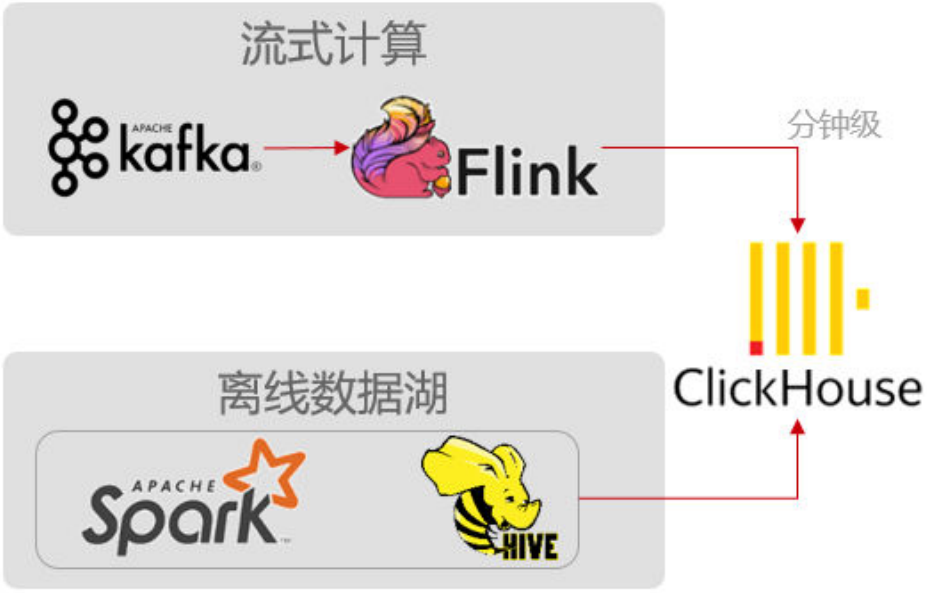
ClickHouse 应用场景

ClickHouse是Click Stream + Data WareHouse的缩写，起初应用于一款Web流量分析工具，基于页面的点击事件流，面向数据仓库进行OLAP分析。当前ClickHouse被广泛的应用于互联网广告、App和Web流量、电信、金融、物联网等众多领域，非常适用于商业智能化应用场景，在全球有大量的应用和实践，具体请参考：**https://clickhouse.tech/docs/en/introduction/adopters/**。

6.2.2 ClickHouse 与其他组件的关系

ClickHouse安装部署依赖ZooKeeper服务。

ClickHouse通过Flink流计算应用加工生成通用的报表数据（明细宽表），准实时写入到ClickHouse，通过Hive/Spark作业加工生成通用的报表数据（明细宽表），批量导入到ClickHouse。



说明

ClickHouse暂不支持对接Kafka普通模式和HDFS安全模式。

6.2.3 ClickHouse 开源增强特性

MRS ClickHouse具备“手动挡”集群模式升级、平滑弹性扩容、高可用HA部署架构等优势能力，具体详情如下：

- 手动挡集群模式升级

  如图6-2所示，多个ClickHouse节点组成的集群，没有中心节点，更多的是一个静态资源池的概念，业务要使用ClickHouse集群模式，需要预先在各个节点的配置文件中定义cluster信息，等所有参与的节点达成共识，业务才可以正确的交互访问，也就是说配置文件中的cluster才是通常理解的“集群”概念。

  图 6-2 ClickHouse 集群

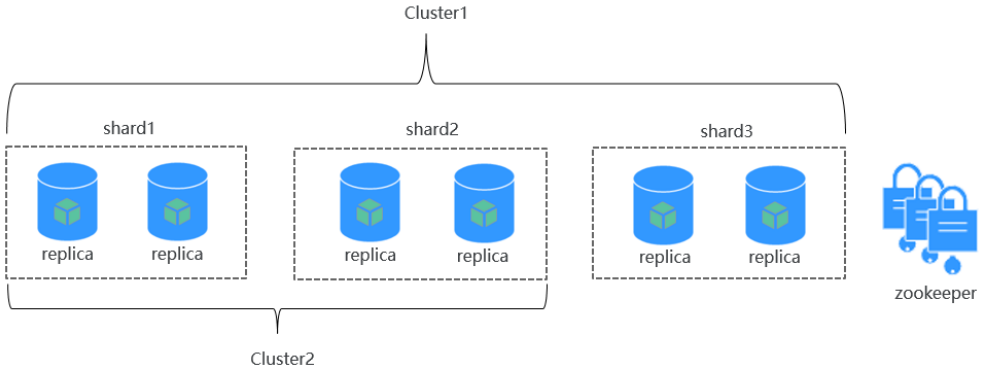
  

  常见的数据库系统，隐藏了表级以下的数据分区、副本存储等细节，用户是无感知的，而ClickHouse则要求用户主动来规划和定义数据分片（shard）、分区（partition）、副本（replica）位置等详细配置。它的这种类似“手动挡”的属

性，给用户带来极不友好的体验，所以MRS服务的ClickHouse实例对这些工作做了统一的打包处理，适配成了“自动挡”，实现了统一管理，灵活易用。

具体部署形态上，一个ClickHouse实例将包含3个ZooKeeper节点和多个ClickHouse节点，采用Dedicated Replica模式，数据双副本高可靠。

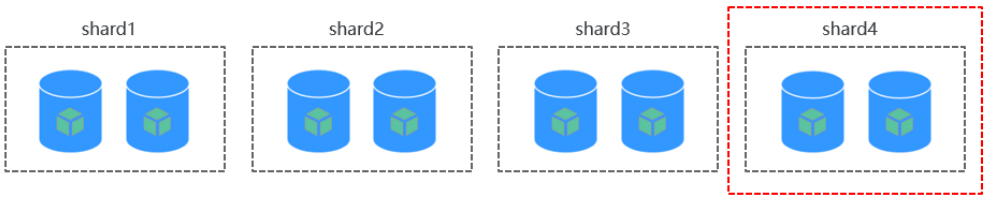
图 6-3 ClickHouse 的 cluster 结构



- 平滑的弹性扩容能力

  随着业务的快速增长，面对集群存储容量或者CPU计算资源接近极限等场景，MRS服务提供了ClickHouse数据迁移工具，该工具可以将某几个ClickHouseServer实例节点上的一个或多个MergeTree引擎分区表的部分分区迁移至其他ClickHouseServer节点上相同的表中，以便保障业务可用性，实现了更加平滑的扩容能力。

  在用户对集群进行扩容ClickHouse节点时，可以使用该工具将原节点上的部分数据迁移至新增节点上，从而达到扩容后的数据均衡。

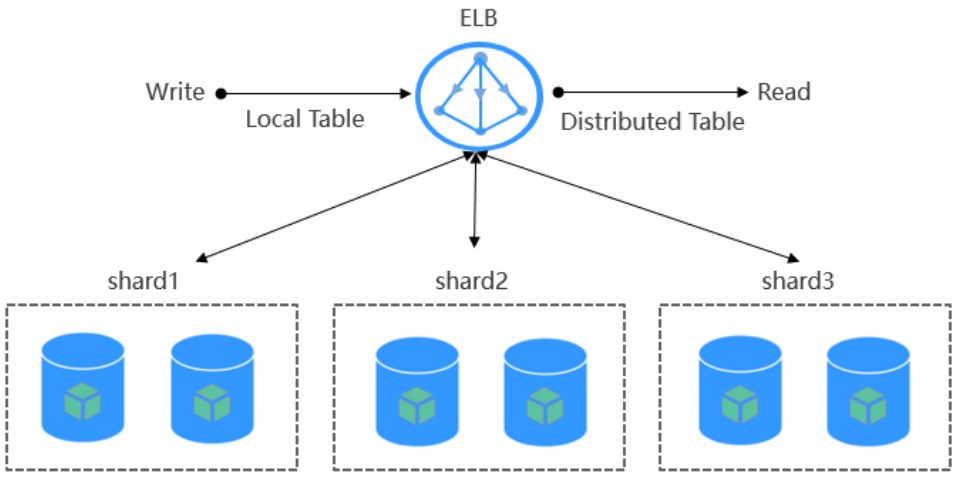


- 高可用HA部署架构

  MRS服务提供了基于ELB的HA部署架构，可以将用户访问流量自动分发到多台后端节点，扩展系统对外的服务能力，实现更高水平的应用容错。如图6-4所示，客户端应用请求集群时，使用ELB（Elastic Load Balance）来进行流量分发，通过ELB的轮询机制，写不同节点上的本地表（Local Table），读不同节点上的分布式表（Distributed Table），这样，无论集群写入的负载、读的负载以及应用接入的高可用性都具备了有力的保障。

  ClickHouse集群发放成功后，每个ClickHouse实例节点对应一个副本replica，两个副本组成一个shard逻辑分片。如创建ReplicatedMergeTree引擎表时，可以指定分片，相同分片内的两个副本数据就可以自动进行同步。

图 6-4 高可用 HA 部署架构图



6.3 CDL

CDL 简介

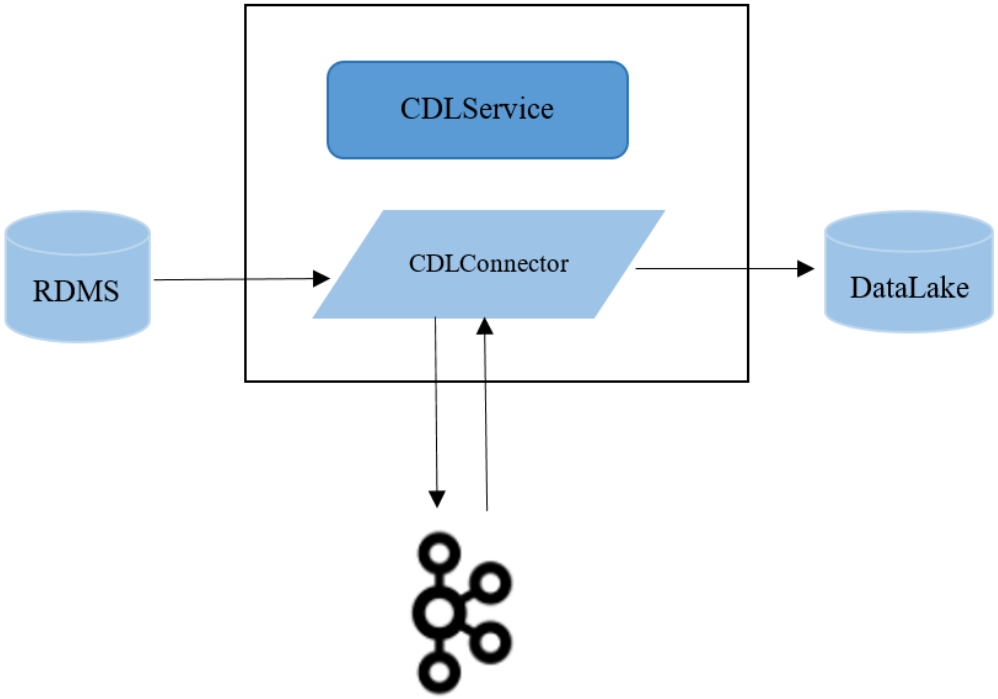
CDL（全称Change Data Loader）是一个基于Kafka Connect框架的实时数据集成服务。 CDL服务能够从各种OLTP数据库中捕获数据库的Data Change事件，并推送到kafka，再由sink connector推送到大数据生态系统中。

CDL目前支持的数据源有MySQL、PostgreSQL、Hudi、Kafka、ThirdParty-Kafka，目标端支持写入Kafka、Hudi、DWS以及ClickHouse。

更多关于CDL组件操作指导，请参考**使用CDL**。

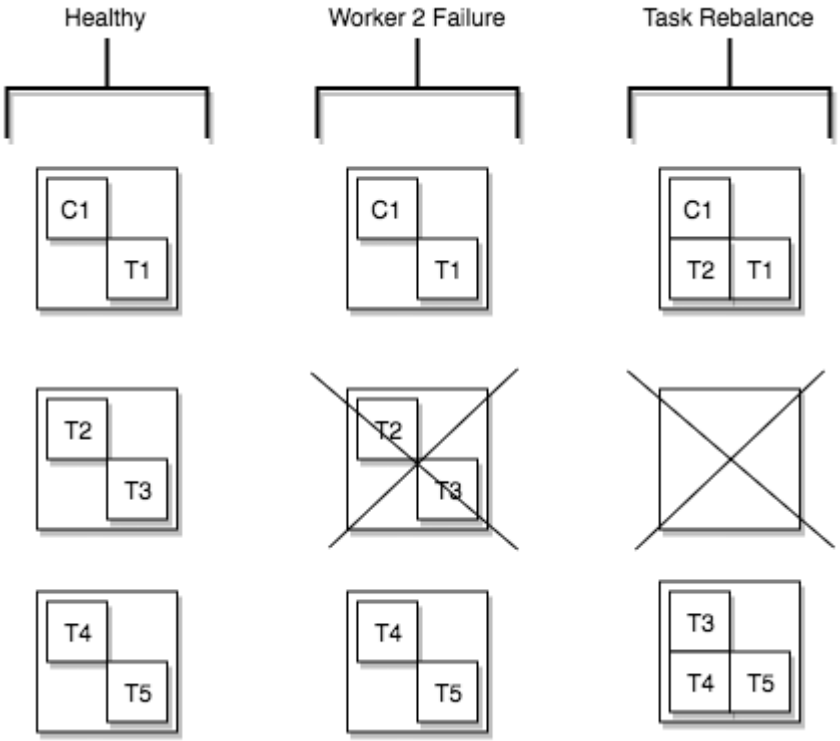
CDL 结构

CDL服务包含了两个重要的角色：CDLConnector和CDLService，CDLConnector是具体执行数据抓取任务的实例，包含了Source Connector和Sink Connector，CDLService是负责管理和创建任务的实例。



CDL服务中的CDLService是多主模式，任意一个CDLService都可以进行业务操作；CDLConnector是分布式模式，提供了高可靠和Rebalance的能力，创建任务时指定的task数量会在整个集群中的CDLConnector实例之间做均衡，保证每个实例上运行的task数量大致相同，如果某个CDLConnector实例异常或者节点宕机，该任务会在其它节点重新平衡task的数量。

图 6-5 Task 的 Rebalance 示意图



CDL 与其他组件的关系

CDL组件基于Kafka Connect框架，抓取的数据都是通过kafka的topic做中转，所以首先依赖Kafka组件，其次CDL本身存储了任务的元数据信息和监控信息，这些数据都存储在数据库，因此也依赖DBService组件。

6.4 DBService

DBService 简介

DBService是一个高可用性的关系型数据库存储系统，适用于存储小量数据（10GB左右），比如：组件元数据。DBService仅提供给集群内部的组件使用，提供数据存储、查询、删除等功能。

DBService是集群的基础组件，Hive、Hue、Oozie、Loader、CDL、Flink、HetuEngine、Kafka、Metadata、Ranger等组件将元数据存储在DBService上，并由DBService提供这些元数据的备份与恢复功能。

更多关于DBService组件操作指导，请参考**使用DBService**。

DBService 结构

DBService组件在集群中采用主备模式部署两个DBServer实例，每个DBServer实例包含三个模块：HA、Database和FloatIP。

其逻辑结构如**图6-6**所示。

图 6-6 DBService 结构

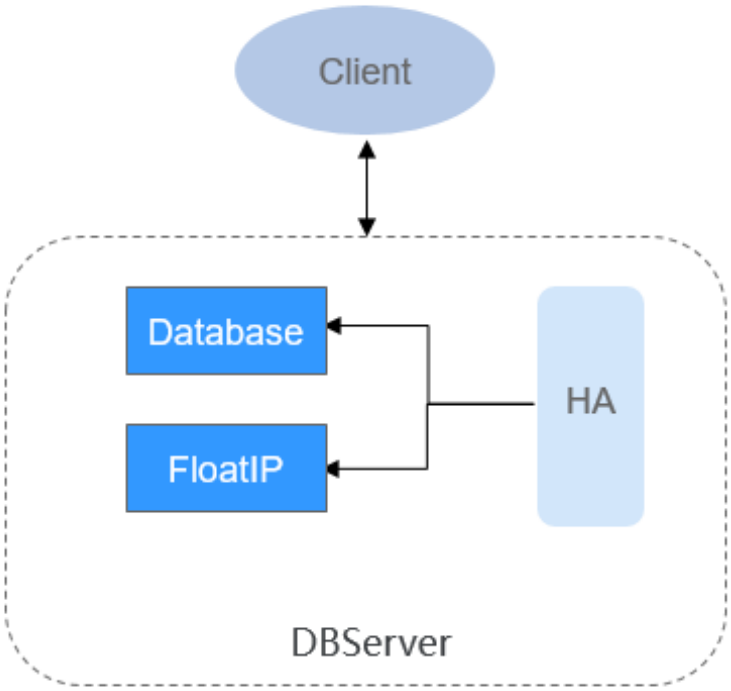


图6-6中各模块的说明如表6-3所示。

表 6-3 模块说明

名称	描述
HA	高可用性管理模块，主备DBServer通过HA进行管理。
Database	数据库模块，存储Client模块的元数据。
FloatIP	浮动IP，对外提供访问功能，只在主DBServer实例上启动浮动IP，Client模块通过该IP访问Database。
Client	使用DBService组件的客户端，部署在组件实例节点上，通过Floatip连接数据库，执行元数据的增加、删除、修改等操作。

DBService 与其他组件的关系

DBService是集群的基础组件，Hive、Hue、Oozie、Loader等组件将元数据存储在DBService上，并由DBService提供这些元数据的备份与恢复功能。

6.5 Doris

6.5.1 Doris 基本原理

Doris 简介

Doris是一个基于MPP架构的高性能、实时的分析型数据库，以极速易用的特点被人们所熟知，仅需亚秒级响应时间即可返回海量数据下的查询结果，不仅可以支持高并发的点查询场景，也能支持高吞吐的复杂分析场景。基于此，Apache Doris能够较好的满足报表分析、即席查询、统一数仓构建、数据湖联邦查询加速等使用场景，用户可以在此之上构建用户行为分析、AB实验平台、日志检索分析、用户画像分析、订单分析等应用。更多相关介绍请参见Apache Doris。

说明

该功能为受限使用功能，需提交工单申请开通。关于Doris组件更多操作指导，请参考使用Doris。

Doris 架构

Doris整体架构如下图所示，FE和BE节点可以横向无限扩展。

图 6-7 Doris 架构

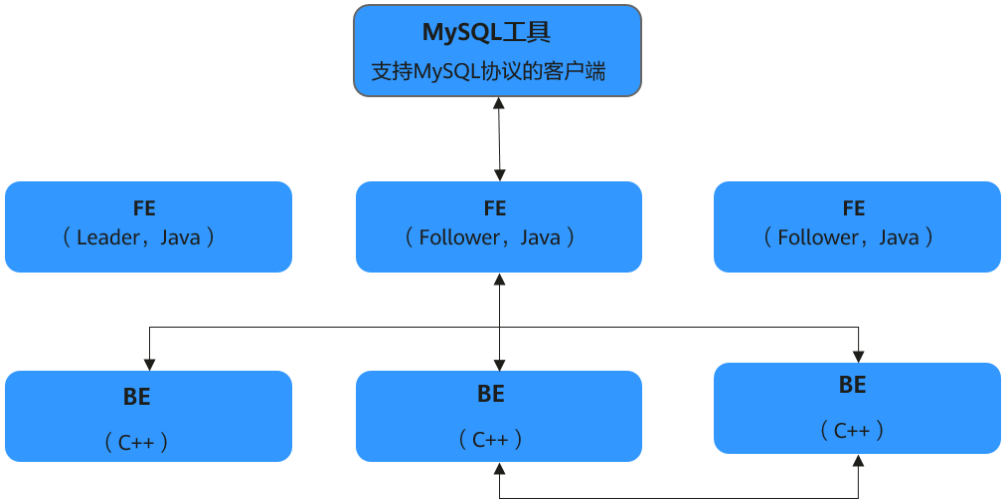


表 6-4 参数说明

名称	说明
MySQL Tools	Doris采用MySQL协议，高度兼容MySQL语法，支持标准SQL，用户可以通过各类客户端工具来访问Doris，并支持与BI工具无缝对接。
FE	主要负责用户请求的接入、查询解析规划、元数据的管理、节点管理相关工作。
BE	主要负责存储数据、执行查询计划、副本负载均衡。
Leader	Leader为Follower组中选举出来的一种角色。
Follower	一条元数据日志需要在多数Follower节点写入成功，才算成功。

Doris采用MPP的模型，节点间和节点内都是并行执行，适用于多个大表的分布式Join。

支持向量化的查询引擎、AQE（ Adaptive Query Execution ）技术、CBO 和 RBO 结合的优化策略、热数据缓存查询等。

Doris 基本概念

在Doris中，数据都以表（Table）的形式进行逻辑上的描述。

- **Row&Column**

  一张表包括行（Row）和列（Column）：

  - Row：即用户的一行数据。
  - Column： 用于描述一行数据中不同的字段。

  Column可以分为两大类：Key和Value。从业务角度看，Key和Value可以分别对应维度列和指标列。从聚合模型的角度来说，Key列相同的行，会聚合成一行。其中Value列的聚合方式由用户在建表时指定。

- **Tablet&Partition**

  在Doris的存储引擎中，用户数据被水平划分为若干个数据分片（Tablet，也称作数据分桶）。每个Tablet包含若干数据行。各个Tablet之间的数据没有交集，并且在物理上是独立存储的。

  多个Tablet在逻辑上归属于不同的分区（Partition）。一个Tablet只属于一个Partition，而一个Partition包含若干个Tablet。因为Tablet在物理上是独立存储的，所以可以视为Partition在物理上也是独立。Tablet是数据移动、复制等操作的最小物理存储单元。

  若干个Partition组成一个Table。Partition可以视为是逻辑上最小的管理单元。数据的导入与删除，只能针对一个Partition进行。

- **数据模型**

  Doris的数据模型主要分为3类：Aggregate、Unique、Duplicate。

  - **Aggregate模型**

    导入数据时，对于Key列相同的行会聚合成一行，而Value列会按照设置的AggregationType进行聚合。 AggregationType目前有以下四种聚合方式：

    - SUM：求和，多行的Value进行累加。
    - REPLACE：替代，下一批数据中的Value会替换之前导入过的行中的Value。
    - MAX：保留最大值。
    - MIN：保留最小值。

  - **Unique模型**

    在某些多维分析场景下，用户更关注的是如何保证Key的唯一性，即如何获得Primary Key唯一性约束。因此，引入了Unique数据模型。

    - **读时合并**

      Unique模型的读时合并实现完全可以用Aggregate模型中的REPLACE方式替代，其内部的实现方式和数据存储方式也完全一样。

    - **写时合并**

      Unique模型的写时合并实现，不同于Aggregate模型，查询性能更接近于Duplicate模型，在有主键约束需求的场景上相比Aggregate模型有较大的查询性能优势，尤其是在聚合查询以及需要用索引过滤大量数据的查询中。

      在开启了写时合并选项的Unique表中，数据在导入阶段就会去将被覆盖和被更新的数据进行标记删除，同时将新的数据写入新的文件。在查询时，所有被标记删除的数据都会在文件级别被过滤，读取出的数据就都是最新的数据，消除了读时合并中的数据聚合过程，并且能够在很多情况下支持多种谓词的下推。因此在许多场景都能带来比较大的性能提升，尤其是在有聚合查询的情况下。

  - **Duplicate模型**

    在某些多维分析场景下，数据既没有主键，也没有聚合需求。可以引入Duplicate数据模型来满足这类需求。

    这种数据模型区别于Aggregate和Unique模型。数据完全按照导入文件中的数据进行存储，不会有任何聚合。即使两行数据完全相同，也都会保留。 而在建表语句中指定的DUPLICATE KEY，只是用来指明底层数据按照指定的列进行排序。

- **数据模型的选择建议**

  因为数据模型在建表时就已经确定，且无法修改。所以，选择一个合适的数据模型非常重要。

  - Aggregate模型可以通过预聚合，极大地降低聚合查询时所需扫描的数据量和查询的计算量，非常适合有固定模式的报表类查询场景。但是该模型对**count(*)**查询不友好。同时因为固定了Value列上的聚合方式，在进行其他类型的聚合查询时，需要考虑语义正确性。
  - Unique模型针对需要唯一主键约束的场景，可以保证主键唯一性约束。但是无法利用ROLLUP等预聚合带来的查询优势。

    Unique模型仅支持整行更新，如果用户既需要唯一主键约束，又需要更新部分列（例如将多张源表导入到一张Doris表的场景），则可以考虑使用Aggregate模型，同时将非主键列的聚合类型设置为REPLACE_IF_NOT_NULL。
  - Duplicate适合任意维度的Ad-hoc查询。虽然同样无法利用预聚合的特性，但是不受聚合模型的约束，可以发挥列存模型的优势（只读取相关列，而不需要读取所有Key列）。

6.5.2 Doris 与其他组件的关系

Doris 与 HDFS 组件的关系

Doris支持导入和导出HDFS数据，并且支持直接查询HDFS数据源。

Doris 与 Hudi 组件的关系

Doris支持直接查询Hudi数据源。

Doris 与 Spark 组件的关系

使用Spark Doris Connector可以通过Spark读取Doris中存储的数据，也支持通过Spark写入数据到Doris。

Doris 与 Flink 组件的关系

使用Flink Doris Connector可以通过Flink操作（读取、插入、修改、删除）Doris中存储的数据。

Doris 与 Hive 组件的关系

Doris支持直接查询Hive数据源。

Doris 与 Kafka 组件的关系

Doris支持导入Kafka的数据。

6.6 Flink

6.6.1 Flink 基本原理

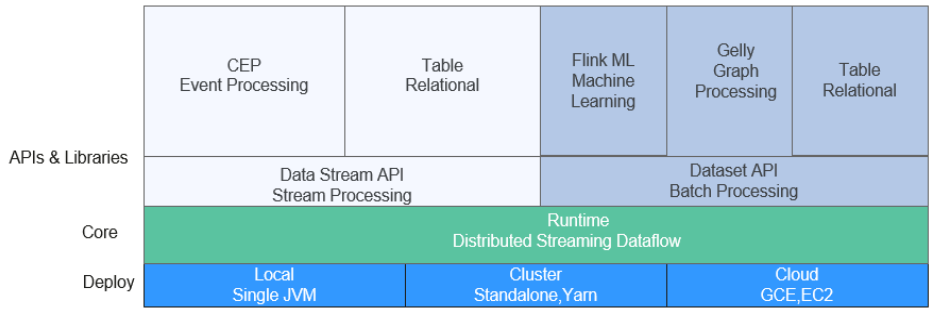
Flink 简介

Flink是一个批处理和流处理结合的统一计算框架，其核心是一个提供了数据分发以及并行化计算的流数据处理引擎。它的最大亮点是流处理，是业界最顶级的开源流处理引擎。

Flink最适合的应用场景是低时延的数据处理（Data Processing）场景：高并发pipeline处理数据，时延毫秒级，且兼具可靠性。

Flink技术栈如图6-8所示。

图 6-8 Flink 技术栈



Flink在当前版本中重点构建如下特性：

- DataStream
- Checkpoint
- 窗口
- Job Pipeline
- 配置表

其他特性继承开源社区，不做增强，具体请参考：**https://ci.apache.org/projects/flink/flink-docs-release-1.12/**。

更多关于Flink组件操作指导，请参考**使用Flink**。

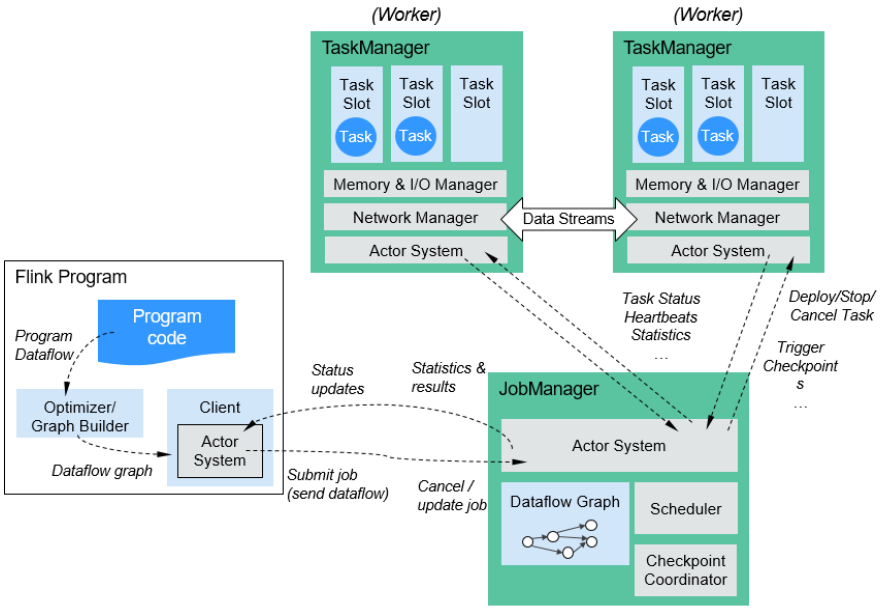
Flink 结构

Flink服务包含了两个重要的角色：FlinkResource和FlinkServer。

- FlinkResource：提供客户端配置管理，是必须安装的角色。包括供客户端下载使用的原始lib包和配置文件，以及FlinkServer提交作业所依赖的原始lib包。无实体进程，作业运行过程不依赖FlinkResource。
- FlinkServer：基于Web的作业管理二次开发平台，可直接在界面开发与管理FlinkSQL作业。具有运维管理界面化、作业开发SQL标准化等特点。

Flink结构如图6-9所示。

图 6-9 Flink 结构



Flink整个系统包含三个部分：

- Client

  Flink Client主要给用户提供向Flink系统提交用户任务（流式作业）的能力。

- TaskManager

  Flink系统的业务执行节点，执行具体的用户任务。TaskManager可以有多个，各个TaskManager都平等。

- JobManager

  Flink系统的管理节点，管理所有的TaskManager，并决策用户任务在哪些TaskManager执行。JobManager在HA模式下可以有多个，但只有一个主JobManager。

如果您想了解更多关于Flink架构的信息，请参考链接：**https://ci.apache.org/projects/flink/flink-docs-master/docs/concepts/flink-architecture/**。

Flink 原理

- **Stream & Transformation & Operator**

  用户实现的Flink程序是由Stream和Transformation这两个基本构建块组成。

  a. Stream是一个中间结果数据，而Transformation是一个操作，它对一个或多个输入Stream进行计算处理，输出一个或多个结果Stream。

  b. 当一个Flink程序被执行的时候，它会被映射为Streaming Dataflow。一个Streaming Dataflow是由一组Stream和Transformation Operator组成，它类似于一个DAG图，在启动的时候从一个或多个Source Operator开始，结束于一个或多个Sink Operator。

  图6-10为一个由Flink程序映射为Streaming Dataflow的示意图。

图 6-10 Flink DataStream 示例

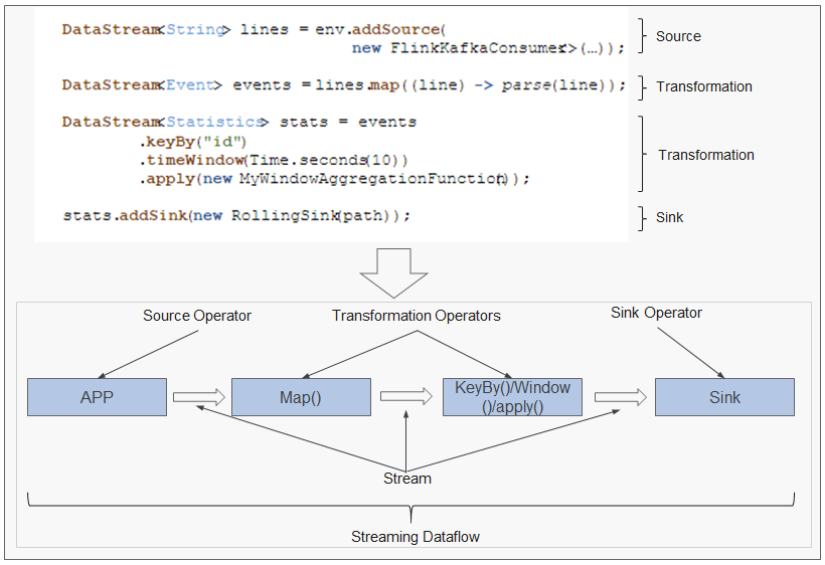


图6-10中“FlinkKafkaConsumer”是一个Source Operator，Map、KeyBy、TimeWindow、Apply是Transformation Operator，RollingSink是一个Sink Operator。

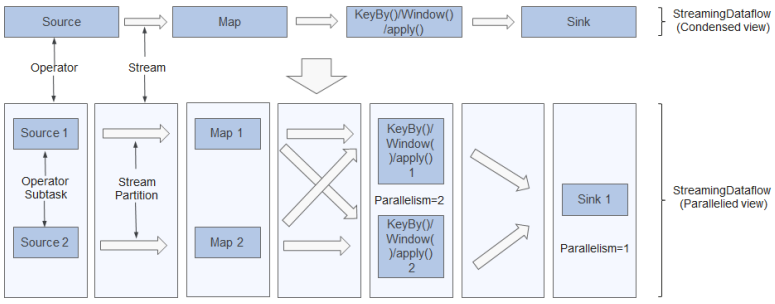
- **Pipeline Dataflow**

  在Flink中，程序是并行和分布式的方式运行。一个Stream可以被分成多个Stream分区（Stream Partitions），一个Operator可以被分成多个Operator Subtask。

  Flink内部有一个优化的功能，根据上下游算子的紧密程度来进行优化。

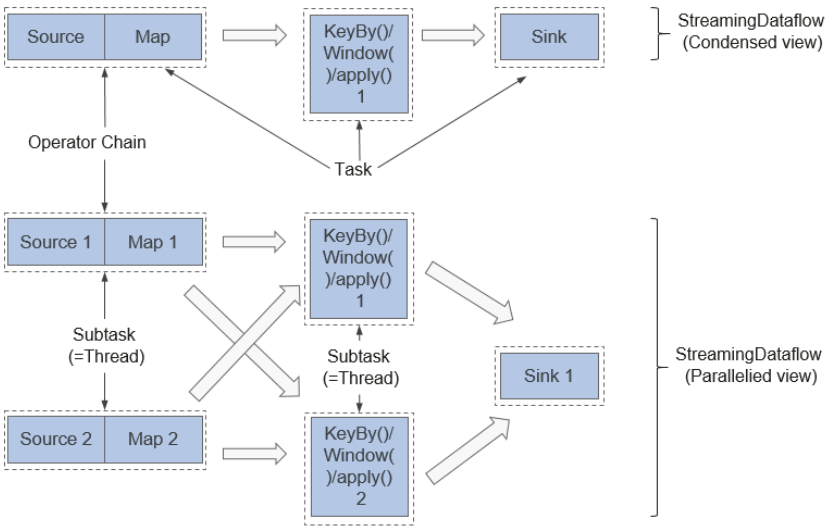
  - 紧密度低的算子则不能进行优化，而是将每一个Operator Subtask放在不同的线程中独立执行。一个Operator的并行度，等于Operator Subtask的个数，一个Stream的并行度（分区总数）等于生成它的Operator的并行度，如图6-11所示。

    图 6-11 Operator

  - 紧密度高的算子可以进行优化，优化后可以将多个Operator Subtask串起来组成一个Operator Chain，实际上就是一个执行链，每个执行链会在TaskManager上一个独立的线程中执行，如图6-12所示。

图 6-12 Operator chain



- 图6-12中上半部分表示的是将Source和Map两个紧密度高的算子优化后串成一个Operator Chain，实际上一个Operator Chain就是一个大的Operator的概念。图中的Operator Chain表示一个Operator，KeyBy表示一个Operator，Sink表示一个Operator，它们通过Stream连接，而每个Operator在运行时对应一个Task，也就是说图中的上半部分有3个Operator对应的是3个Task。
- 图6-12中下半部分是上半部分的一个并行版本，对每一个Task都并行化为多个Subtask，这里只是演示了2个并行度，Sink算子是1个并行度。

Flink 关键特性

- 流式处理

  高吞吐、高性能、低时延的实时流处理引擎，能够提供毫秒级时延处理能力。
- 丰富的状态管理

  流处理应用需要在一定时间内存储所接收到的事件或中间结果，以供后续某个时间点访问并进行后续处理。Flink提供了丰富的状态管理相关的特性，包括：
  - 多种基础状态类型：Flink提供了多种不同数据结构的状态支持，如ValueState、ListState、MapState等。用户可以基于业务模型选择最高效、合适状态类型。
  - 丰富的State Backend：State Backend负责管理应用程序的状态，并根据需要进行Checkpoint。Flink提供了不同State Backend，State可以存储在内存上或RocksDB等上，并支持异步以及增量的Checkpoint机制。
  - 精确一次语义：Flink的Checkpoint和故障恢复能力保证了任务在故障发生前后的应用状态一致性，为某些特定的存储支持了事务型输出的功能，即使在发生故障的情况下，也能够保证精确一次的输出。
- 丰富的时间语义

  时间是流处理应用的重要组成部分，对于实时流处理应用来说，基于时间语义的窗口聚合、检测、匹配等运算是很常见的。Flink提供了丰富的时间语义。
  - Event-time：使用事件本身自带的时间戳进行计算，使乱序到达或延迟到达的事件处理变得更加简单。

  - Watermark：Flink引入Watermark概念，用以衡量事件时间的发展。Watermark也为平衡处理时延和数据完整性提供了灵活的保障。当处理带有Watermark的事件流时，在计算完成之后仍然有相关数据到达时，Flink提供了多种处理选项，如将数据重定向（side output）或更新之前完成的计算结果。
  - Processing-time和Ingestion-time。
  - 高度灵活的流式窗口：Flink能够支持时间窗口、计数窗口、会话窗口，以及数据驱动的自定义窗口，可以通过灵活的触发条件定制，实现复杂的流式计算模式。

- 容错机制

  分布式系统，单个Task或节点的崩溃或故障，往往会导致整个任务的失败。Flink提供了任务级别的容错机制，保证任务在异常发生时不会丢失用户数据，并且能够自动恢复。

  - Checkpoint：Flink基于Checkpoint实现容错，用户可以自定义对整个任务的Checkpoint策略，当任务出现失败时，可以将任务恢复到最近一次Checkpoint的状态，从数据源重发快照之后的数据。
  - Savepoint：一个Savepoint就是应用状态的一致性快照，Savepoint与Checkpoint机制相似，但Savepoint需要手动触发，Savepoint保证了任务在升级或迁移时，不丢失当前流应用的状态信息，便于任何时间点的任务暂停和恢复。

- Flink SQL

  Table API和SQL借助了Apache Calcite来进行查询的解析，校验以及优化，可以与DataStream和DataSet API无缝集成，并支持用户自定义的标量函数，聚合函数以及表值函数。简化数据分析、ETL等应用的定义。下面代码示例展示了如何使用Flink SQL语句定义一个会话点击量的计数应用。

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

  有关Flink SQL的更多信息，请参见：**https://ci.apache.org/projects/flink/flink-docs-master/dev/table/sqlClient.html**。

- CEP in SQL

  Flink允许用户在SQL中表示CEP（Complex Event Processing）查询结果以用于模式匹配，并在Flink上对事件流进行评估。

  CEP SQL通过MATCH_RECOGNIZE的SQL语法实现。MATCH_RECOGNIZE子句自Oracle Database 12c起由Oracle SQL支持，用于在SQL中表示事件模式匹配。CEP SQL使用举例如下：

  ```
  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 Flink HA 方案介绍

Flink HA 方案介绍

每个Flink集群只有单个JobManager，存在单点失败的情况。Flink有Yarn、Standalone和Local三种模式，其中Yarn和Standalone是集群模式，Local是指单机模式。但Flink对于Yarn模式和Standalone模式提供HA机制，使集群能够从失败中恢复。这里主要介绍Yarn模式下的HA方案。

Flink支持HA模式和Job的异常恢复。这两项功能高度依赖ZooKeeper，在使用之前用户需要在“flink-conf.yaml”配置文件中配置ZooKeeper，配置ZooKeeper的参数如下：

```
high-availability: zookeeper
high-availability.zookeeper.quorum: ZooKeeperIP地址:2181
high-availability.storageDir: hdfs:///flink/recovery
```

Yarn模式

Flink的JobManager与Yarn的Application Master（简称AM）是在同一个进程下。Yarn的ResourceManager对AM有监控，当AM异常时，Yarn会将AM重新启动，启动后，所有JobManager的元数据从HDFS恢复。在恢复期间，旧的业务不能运行，新的业务不能提交。ZooKeeper上还是存有JobManager的元数据，比如运行Job的信息，会提供给新的JobManager使用。对于TaskManager的失败，由JobManager上Akka的DeathWatch机制处理。当TaskManager失败后，重新向Yarn申请容器，创建TaskManager。

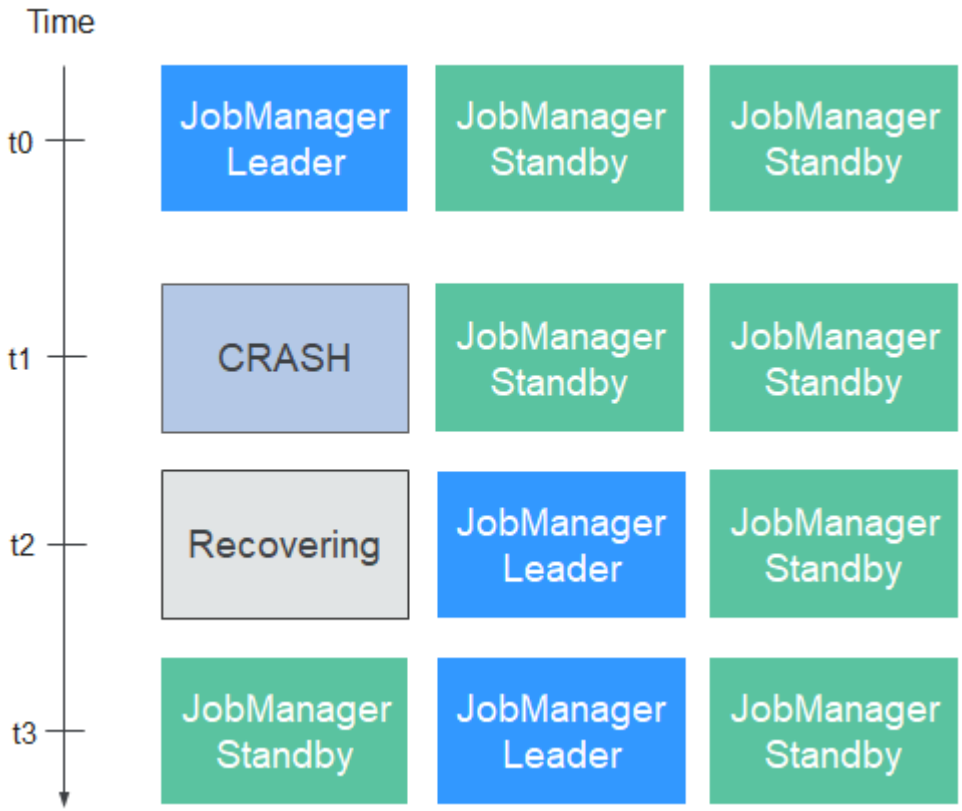
Yarn模式的HA方案的更多信息，可参考链接：

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

Standalone模式

对于Standalone模式的集群，可以启动多个JobManager，然后通过ZooKeeper选举出Leader作为实际使用的JobManager。该模式下可以配置一个主JobManager（Leader JobManager）和多个备JobManager（Standby JobManager），这能够保证当主JobManager失败后，备的某个JobManager可以承担主的职责。**图6-13**为主备JobManager的恢复过程。

图 6-13 恢复过程



TaskManager恢复

对于TaskManager的失败，由JobManager上Akka的DeathWatch机制处理。当TaskManager失败后，由JobManager负责创建一个新TaskManager，并把业务迁移到新的TaskManager上。

JobManager恢复

Flink的JobManager与Yarn的Application Master（简称AM）是在同一个进程下。Yarn的ResourceManager对AM有监控，当AM异常时，Yarn会将AM重新启动，启动后，所有JobManager的元数据从HDFS恢复。但恢复期间，旧的业务不能运行，新的业务不能提交。

Job恢复

Job的恢复必须在Flink的配置文件中配置重启策略。当前包含三种重启策略：fixed-delay、failure-rate和none。只有配置fixed-delay、failure-rate，Job才可以恢复。另外，如果配置了重启策略为none，但Job设置了Checkpoint，默认会将重启策略改为fixed-delay，且重试次数是配置项“restart-strategy.fixed-delay.attempts”的值。

三种策略的具体信息请参考Flink官网：**https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/task_failure_recovery.html**。配置策略的参考如下：

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

以下场景的异常，都会导致Job重新恢复：

- 当JobManager失败后，所有Job会停止，直到新的JobManager运行后，所有Job恢复。
- 当某一TaskManager失败后，这个TaskManager上的所有作业都将停止，然后等待有可用资源后重启。
- 当某个Job的Task失败后，整个Job也会重启。

说明

有关Job的配置重启策略，具体内容请参见https://ci.apache.org/projects/flink/flink-docs-release-1.12/ops/jobmanager_high_availability.html。

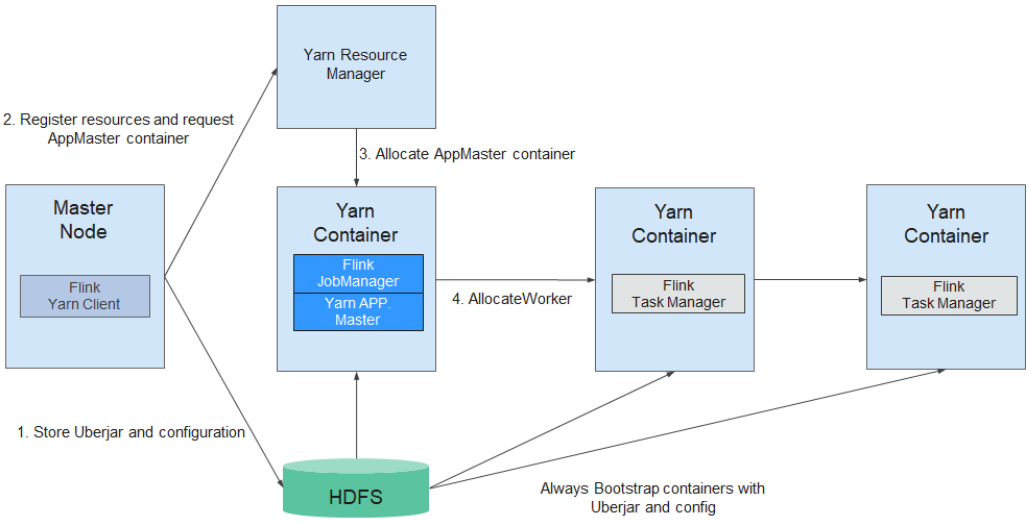
6.6.3 Flink 与其他组件的关系

Flink 与 Yarn 的关系

Flink支持基于Yarn管理的集群模式，在该模式下，Flink作为Yarn上的一个应用，提交到Yarn上执行。

Flink基于Yarn的集群部署如图6-14所示。

图 6-14 Flink 基于 Yarn 的集群部署



1. Flink Yarn Client首先会检验是否有足够的资源来启动Yarn集群，如果资源足够，会将Jar包、配置文件等上传到HDFS。
2. Flink Yarn Client首先与Yarn Resource Manager进行通信，申请启动Application Master（以下简称AM）的Container，并启动AM。等所有的Yarn的Node Manager将HDFS上的Jar包、配置文件下载后，则表示AM启动成功。
3. AM在启动的过程中会和Yarn的RM进行交互，向RM申请需要的Task Manager Container，申请到Task Manager Container后，启动TaskManager进程。
4. 在Flink Yarn的集群中，AM与Flink JobManager在同一个Container中。AM会将JobManager的RPC地址通过HDFS共享的方式通知各个TaskManager，TaskManager启动成功后，会向JobManager注册。
5. 等所有TaskManager都向JobManager注册成功后，Flink基于Yarn的集群启动成功，Flink Yarn Client就可以提交Flink Job到Flink JobManager，并进行后续的映射、调度和计算处理。

6.6.4 Flink 滑动窗口增强

本节主要介绍Flink滑动窗口以及滑动窗口的优化方式。

Flink窗口的详细内容请参见官网：**https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/stream/operators/windows.html**。

窗口介绍

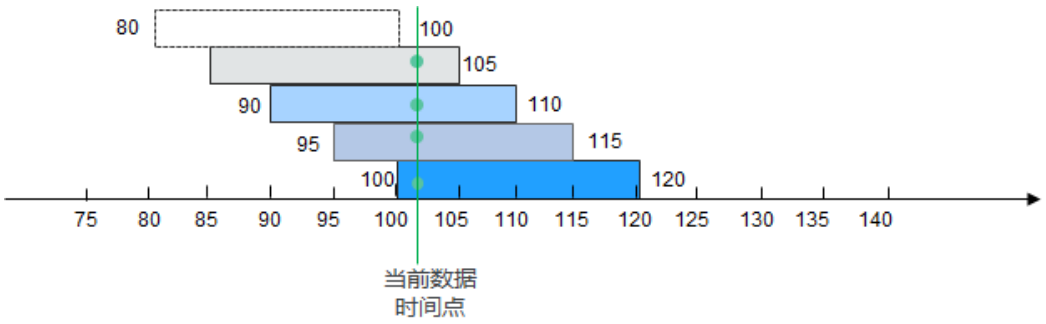
窗口中数据的保存形式主要有中间结果和原始数据两种，对窗口中的数据使用公共算子，如sum等操作时（window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).sum）仅会保留中间结果；当用户使用自定义窗口时（window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds(5))).apply(new UDF)）保存所有的原始数据。

用户使用自定义SlidingEventTimeWindow和SlidingProcessingTimeWindow时，数据以多备份的形式保存。假设窗口的定义如下：

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

当一个数据到来时，会被分配到20/5=4个不同的窗口中，即数据在内存中保存了4份。当窗口大小/滑动周期非常大时，冗余现象非常严重。

图 6-15 窗口原始结构示例



假设一个数据在102秒时到来，它将会被分配到[85, 105)、[90, 110)、[95, 115)以及[100, 120)四个不同的窗口中。

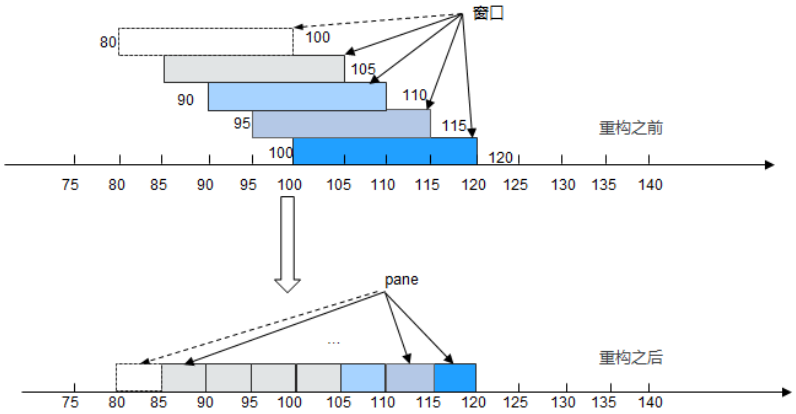
窗口优化

针对上述SlidingEventTimeWindow和SlidingProcessingTimeWindow在保存原始数据时存在的数据冗余问题，对保存原始数据的窗口进行重构，优化存储，使其存储空间大大降低，具体思路如下：

1. 以滑动周期为单位，将窗口划分为若干相互不重合的pane。

   每个窗口由一到多个pane组成，多个pane对窗口构成了覆盖关系。所谓一个pane即一个滑动周期，如：在窗口window(SlidingEventTimeWindows.of(Time.seconds(20), Time.seconds.of(5)))中pane的大小为5秒，假设这个窗口为[100, 120)，则包含的pane为[100, 105), [105, 110), [110, 115), [115, 120)。

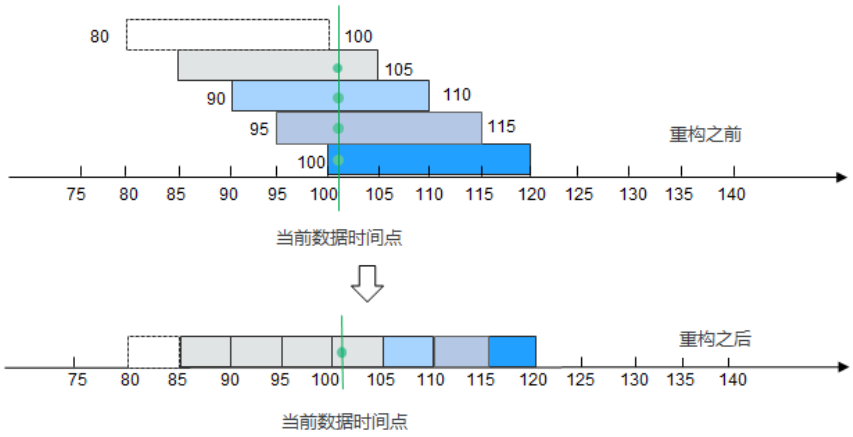
图 6-16 窗口重构示例



2. 当某个数据到来时，并不分配到具体的窗口中，而是根据自己的时间戳计算出该数据所属的pane，并将其保存到对应的pane中。

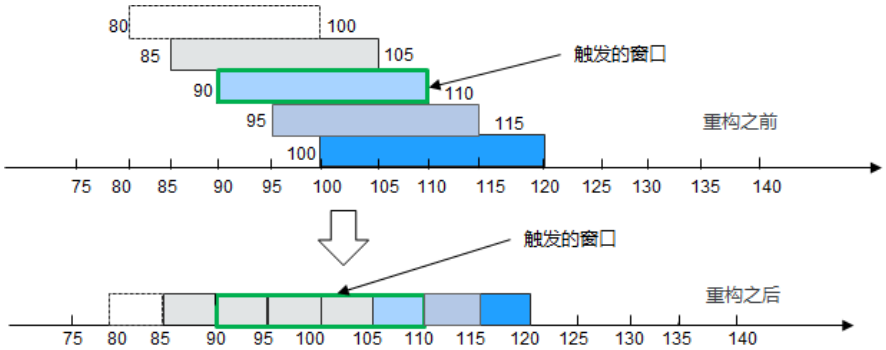
   一个数据仅保存在一个pane中，内存中只有一份。

图 6-17 窗口保存数据示例



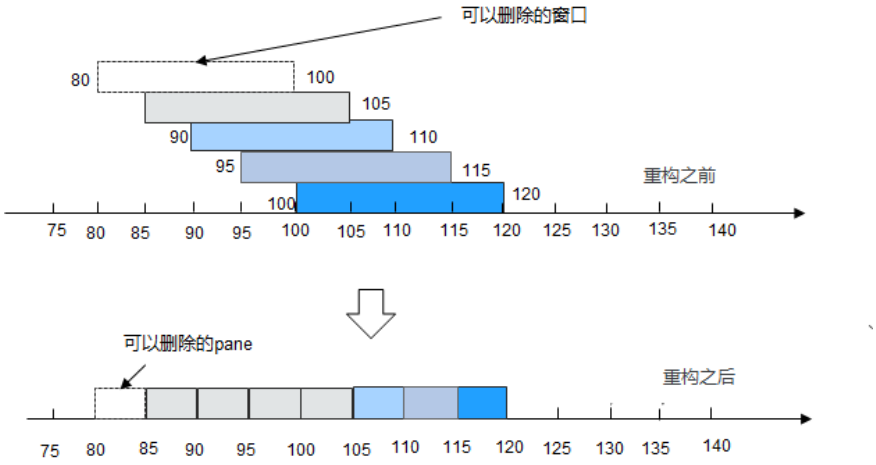
3. 当需要触发某个窗口时，计算该窗口包含的所有pane，并取出合并成一个完整的窗口计算。

图 6-18 窗口触发计算示例



4. 当某个pane不再需要时，将其从内存中删除。

图 6-19 窗口删除示例



通过优化，可以大幅度降低数据在内存以及快照中的数量。

6.6.5 Flink Job Pipeline 增强

通常情况下，开发者会将与某一方面业务相关的逻辑代码放在一个比较大的Jar包中，这种Jar包称为Fat Jar。

Fat Jar具有以下缺点：

- 随着业务逻辑越来越复杂，Jar包的大小也不断增加。
- 协调难度增大，所有的业务开发人员都在同一套业务逻辑上开发，虽然可以将整个业务逻辑划分为几个模块，但各模块之间是一种紧耦合的关系，当需求更改时，需要重新规划整个流图。

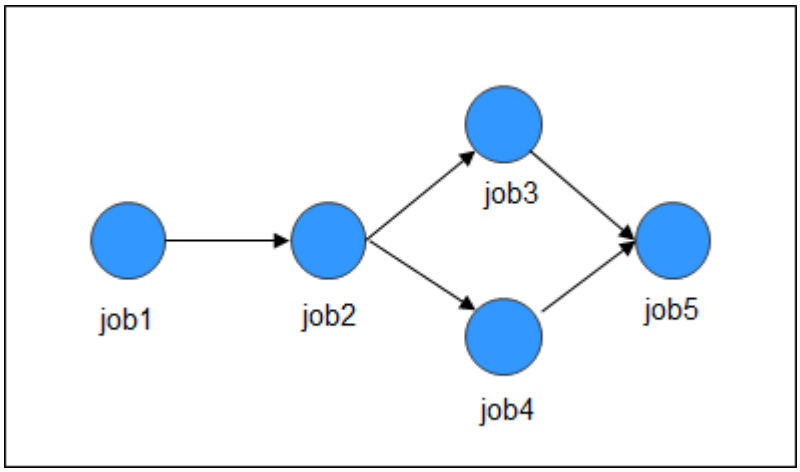
拆分成多个作业目前还存在问题。

- 通常情况下，作业之间可以通过Kafka实现数据传输，如作业A可以将数据发送到Kafka的Topic A下，然后作业B和作业C可以从Topic A下读取数据。该方案简单易行，但是延迟一般大于100ms。
- 采用TCP直接相连的方式，算子在分布式环境下，可能会调度到任意节点，上下游之间无法感知其存在。

Job Pipeline流图结构

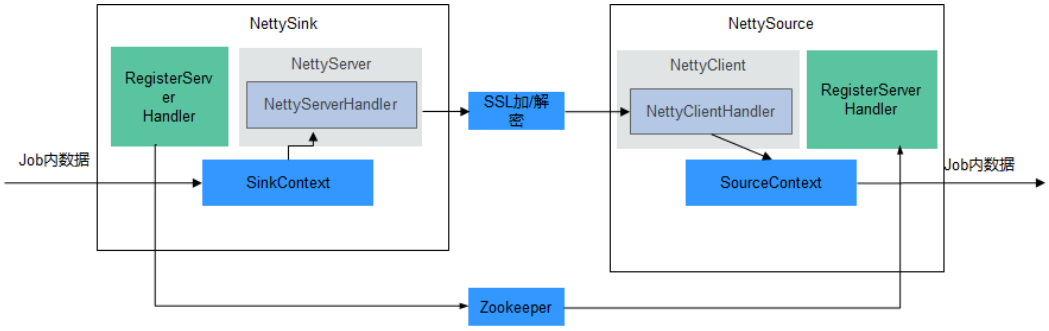
Pipeline是由Flink的多个Job通过TCP连接起来，上游Job可以直接向下游Job发送数据。这种发送数据的流图称为Job Pipeline，如图6-20所示。

图 6-20 Job Pipeline 流图



Job Pipeline原理介绍

图 6-21 Job Pipeline



- NettySink和NettySource

  Pipeline中上下游Job是直接通过Netty进行通信，上游Job的Sink算子作为Server，下游Job的Source算子作为Client。上游Job的Sink算子命名为NettySink，下游Job的Source算子命名为NettySource。
- NettyServer和NettyClient

  NettySink作为Netty的服务器端，内部NettyServer实现服务器功能；NettySource作为Netty的客户端，内部NettyClient实现客户端功能。
- 发布者

  通过NettySink向下游Job发送数据的Job称为发布者。
- 订阅者

  通过NettySource接收上游Job发送的数据的Job称为订阅者。
- 注册服务器

  保存NettyServer的IP、端口以及NettySink的并发度信息的第三方存储器。
- 总体架构是一个三层结构，由外到里依次是：
  - NettySink->NettyServer->NettyServerHandler
  - NettySource->NettyClient->NettyClientHandler

Job Pipeline功能介绍

- **NettySink**

  NettySink由以下几个重要模块组成：

  - RichParallelSinkFunction

    NettySink继承了RichParallelSinkFunction，使其具有Sink算子的属性。主要通过RichParallelSinkFunction的接口来实现以下功能：

    - 启动NettySink算子。
    - 运行NettySink算子，从本Job的上游算子接收数据。
    - 取消NettySink算子运行等。

    也可以通过其属性获取以下信息：

    - NettySink算子各个并发度的subtaskIndex信息。
    - NettySink算子的并发度。

  - RegisterServerHandler

    该组件主要是与注册服务器交互的部件，在平台上定义了一系列接口，包括以下几种接口：

    - “start();”：启动RegisterServerHandler，与第三方RegisterServer建立联系。
    - “createTopicNode();”：创建Topic节点。
    - “register();”： 将IP、端口及并发度信息注册到Topic节点下。
    - “deleteTopicNode();”： 删除Topic节点。
    - “unregister();”： 删除注册信息。
    - “query(); ”：查询注册信息。
    - “isExist();”： 查找某个信息是否存在。
    - “shutdown(); ”：关闭RegisterServerHandler，与第三方RegisterServer断开连接。

说明

- RegisterServerHandler接口实现了ZooKeeper作为RegisterServer的Handler，用户可以根据自己的需求，实现自己的Handler，ZooKeeper中信息的保存形式如下图所示：

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

- Namespace的信息通过“flink-conf.yaml”的以下配置项获取：

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

- ZookeeperRegisterServerHandler与ZooKeeper之间的SASL认证通过Flink的框架实现。
- 用户必须自己保证每个Job有一个唯一的TOPIC，否则会引起作业间订阅关系的混乱。
- 在ZookeeperRegisterServerHandler调用shutdown()时，首先删除本并发度的注册信息，然后尝试删除TOPIC节点，如果TOPIC节点为非空，则放弃删除TOPIC节点，说明其他并发度还未退出。

– NettyServer

该模块是NettySink算子的核心之一，主要作用是创建一个NettyServer并接收NettyClient的连接申请。将同一Job中上游算子发送过来的数据，经由NettyServerHandler发送出去。 另外，NettyServer的端口及子网需要在“flink-conf.yaml”配置文件中配置：

- 端口范围

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

- 子网

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

说明

nettyconnector.sinkserver.subnet默认配置为Flink客户端所在节点子网，若客户端与TaskManager不在同一个子网则有可能导致错误，需手动配置为TaskManager所在网络子网（业务IP）。

– NettyServerHandler

该Handler是NettySink与订阅者交互的通道，当NettySink接收到消息时，该Handler负责将消息发送出去。为保证数据传输的安全性，该通道通过SSL加密。另外设置一个Netty Connector的功能开关，只有当Flink的SSL总开关被打开以及配置“nettyconnector.ssl.enabled”为“true”的时候才开启SSL加密，否则不开启。

- **NettySource**

NettySource由以下几个重要模块组成：

– RichParallelSourceFunction

NettySource继承了RichParallelSinkFunction，使其具有Source算子的属性，主要通过RichParallelSourceFunction接口来实现以下功能：

- 启动NettySink算子。
- 运行NettySink算子，接收来自订阅者的数据并注入到所在Job中。
- 取消Source算子运行等。

也可以通过其属性获取以下信息：

- NettySource算子各个并发度的subtaskIndex信息。
- NettySource算子的并发度。

当NettySource算子进入run阶段后，平台内部会不断监控其NettyClient状态是否健康，一旦发现其出现异常，即会重启NettyClient，重新与NettyServer建立连接并接收数据，以防接收的数据混乱。

- RegisterServerHandler

  该组件与NettySink的RegisterServerHandler功能相同，在NettySource算子中仅获取所订阅Job的各个并发算子的IP、端口及并发算子信息。

- NettyClient

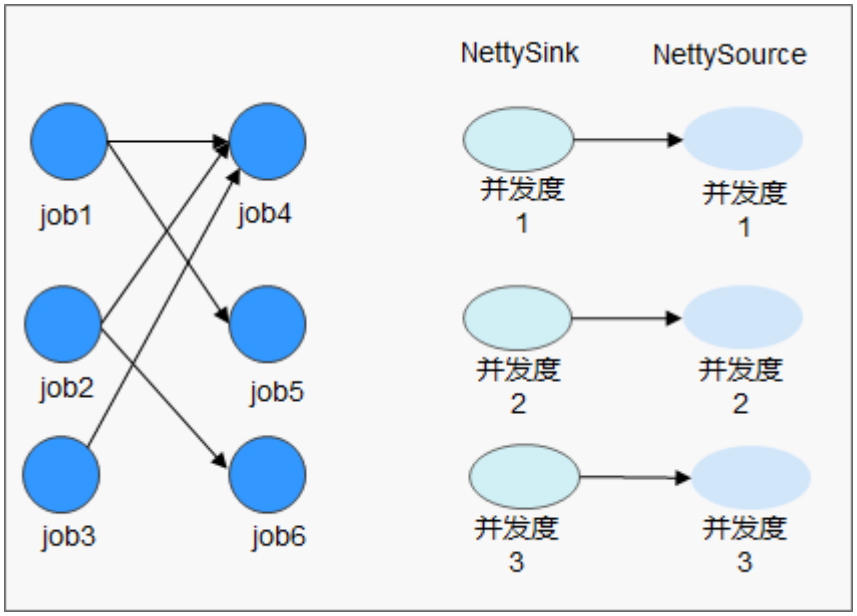
  NettyClient与NettyServer建立连接，并通过NettyClientHandler接收数据。每个NettySource算子必须具有唯一的name（由用户来保障）。NettyServer通过唯一的name确定每个Client来自不同的NettySource。当NettyClient与NettyServer建立连接时，首先向NettyServer注册NettyClient，将NettyClient的NettySource name传递给NettyServer。

- NettyClientHandler

  该模块是与发布者交互的通道，也是与Job的其他算子交互的通道。当该通道中接收到消息时，该Handler负责将消息注入到Job内部。另外，为保证数据安全传输，该通道通过SSL加密，与NettySink进行通信。另外设置一个NettyConnector的功能开关，只有当Flink的SSL总开关被打开以及“nettyconnector.ssl.enabled”为“true”的时候才开启SSL加密，否则不开启。

Job与Job之间的联系可能是多对多的关系，对于每个NettySink和NettySource算子的并发度而言，是一对多的关系，如**图6-22**所示。

图 6-22 关系图



6.6.6 Flink Stream SQL Join 增强

Flink的Table API&SQL是一种用于Scala和Java的语言集成式查询API，它支持非常直观的从关系运算符（如选择、筛选和连接）进行组合查询。Table API&SQL详细内容请参见官网：**https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/table/index.html**。

Stream SQL Join介绍

SQL Join用于根据两个或多个表中的列之间的关系，从这些表中查询数据。Flink Stream SQL Join允许对两个流式table进行Join，并从中查询结果。支持类似于以下内容的查询：

```
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;
```

目前，Stream SQL Join需在指定的窗口范围内进行。对窗口范围内的数据进行连接，需要至少一个相等连接谓词和一个绑定双方时间的条件。这个条件可以由两个适当的范围谓词（<、<=、>=、>），一个**BETWEEN**谓词或者一个单一的相等谓词来定义。这个相等谓词主要是比较两个输入表的同类型时间属性（比如处理时间或者事件时间）。

以下是一个关于在收到订单后四小时内发货， 将所有订单及其相应的货件进行Join的示例：

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

说明

1. Stream SQL Join仅支持Inner Join。
2. ON子句应包括相等连接条件。
3. 时间属性只支持处理时间和事件时间。
4. 窗口条件只支持有界的时间范围，如**o.proctime BETWEEN s.proctime - INTERVAL '1' HOUR AND s.proctime + INTERVAL '1' HOUR**，不支持像**o. proctime > s.proctime**这样无界的范围，并应包括两个流的proctime属性，不支持**o.proctime BETWEEN proctime () AND proctime () + 1**。

6.6.7 Flink CEP in SQL 增强

SQL 中的 Flink CEP

CloudStream扩展为允许用户在SQL中表示CEP查询结果以用于模式匹配，并在Flink引擎上对事件流进行评估。

SQL 查询语法

通过MATCH_RECOGNIZE的SQL语法实现。MATCH_RECOGNIZE子句自Oracle Database 12c起由Oracle SQL支持，用于在SQL中表示事件模式匹配。Apache Calcite同样支持MATCH_RECOGNIZE子句。

由于Flink通过Calcite分析SQL查询结果，本操作遵循Apache Calcite语法。

```
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 ]*
    )
```

MATCH_RECOGNIZE子句的语法元素定义如下：

-PARTITION BY [可选]：定义分区列。该子句为可选子句。如果未定义，则使用并行度1。

-ORDER BY [可选]：定义数据流中事件的顺序。ORDER BY子句为可选子句，如果忽略则使用非确定性排序。由于事件顺序在模式匹配中很重要，因此大多数情况下应指定该子句。

-MEASURES [可选]：指定匹配成功的事件的属性值。

-ONE ROW PER MATCH | ALL ROWS PER MATCH [可选]：定义如何输出结果。ONE ROW PER MATCH表示每次匹配只输出一行，ALL ROWS PER MATCH表示每次匹配的每一个事件输出一行。

-AFTER MATCH [可选]：指定从何处开始对下一个模式匹配进行匹配成功后的处理。

-PATTERN： 将匹配模式定义为正则表达式格式。PATTERN子句中可使用以下运算符： 连接运算符，量词运算符(*，+，?，{n}，{n,}, {n,m}, {,m}),分支运算符（使用竖线‘|’），以及异运算符（ ‘{- -}’）。

-WITHIN [可选]：当且仅当匹配发生在指定时间内，则输出模式子句匹配。

-SUBSET [可选]：将DEFINE子句中定义的一个或多个关联变量组合在一起。

-DEFINE：指定boolean条件，该条件定义了PATTERN子句中使用的变量。

此外，还支持以下函数：

-MATCH_NUMBER()：可用于MEASURES子句中，为同一成功匹配的每一行分配相同编号。

-CLASSIFIER()：可用于MEASURES子句中，以指示匹配的行与变量之间的映射关系。

-FIRST()和LAST()：可用于MEASURES子句中，返回在映射到模式变量的行集的第一行或最后一行中评估的表达式的值。

-NEXT()和PREV()：可用于DEFINE子句中，通过分区中的前一行或下一行来评估表达式。

-RUNNING和FINAL关键字：可用于确定聚合的所需语义。RUNNING可用于MEASURES和DEFINE子句中，而FINAL只能用于MEASURES子句中。

-聚合函数(COUNT，SUM，AVG，MAX，MIN)：这些聚合函数可用于MEASURES子句和DEFINE子句中。

查询示例

以下查询发现股票价格数据流中的V型模式。

```
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)
   )
```

在以下查询中，聚合函数AVG应用于A和C相关变量组成的SUBSET E的MEASURES子句中。

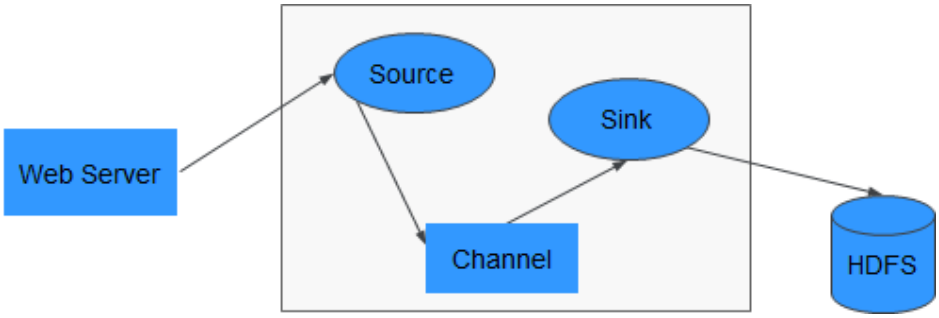
```
SELECT *
   FROM Ticker
   MATCH_RECOGNIZE (
     MEASURES
       AVG(E.price) AS avgPrice
     ONE ROW PER MATCH
     AFTER MATCH SKIP PAST LAST ROW
     PATTERN (A B+ C)
     SUBSET E = (A,C)
     DEFINE
       A AS A.price < 30,
       B AS B.price < 20,
       C AS C.price < 30
   )
```


6.7 Flume

6.7.1 Flume 基本原理

Flume是一个高可用、高可靠，分布式的海量日志采集、聚合和传输的系统。Flume支持在日志系统中定制各类数据发送方，用于收集数据；同时，Flume提供对数据进行简单处理，并写到各种数据接收方（可定制）的能力。其中Flume-NG是Flume的一个分支，其特点是明显简单，体积更小，更容易部署，其最基本的架构如下图所示：

图 6-23 Flume-NG 架构



Flume-NG由一个个Agent来组成，而每个Agent由Source、Channel、Sink三个模块组成，其中Source负责接收数据，Channel负责数据的传输，Sink则负责数据向下一端的发送。

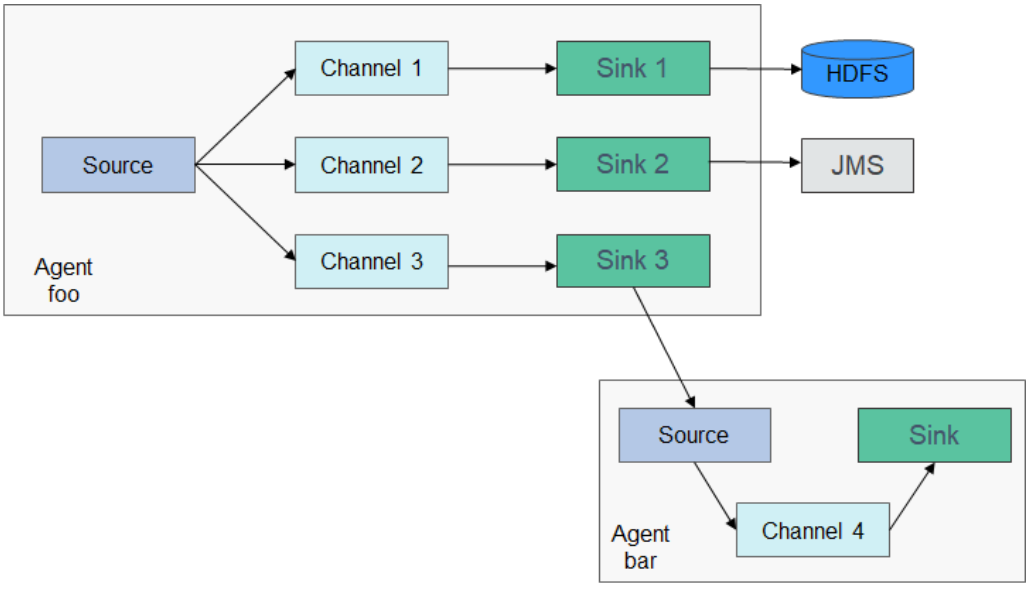
表 6-5 模块说明

名称	说明
Source	Source负责接收数据或通过特殊机制产生数据，并将数据批量放到一个或多个Channel。Source的类型有数据驱动和轮询两种。 典型的Source类型如下： ● 和系统集成的Sources：Syslog、Netcat。 ● 自动生成事件的Sources：Exec、SEQ。 ● 用于Agent和Agent之间通信的IPC Sources：Avro。 Source必须至少和一个Channel关联。

名称	说明
Channel	Channel位于Source和Sink之间，用于缓存来自Source的数据，当Sink成功将数据发送到下一跳的Channel或最终目的地时，数据从Channel移除。 Channel提供的持久化水平与Channel的类型相关，有以下三类： ● Memory Channel：非持久化。 ● File Channel：基于WAL（预写式日志Write-Ahead Logging）的持久化实现。 ● JDBC Channel：基于嵌入Database的持久化实现。 Channel支持事务，可提供较弱的顺序保证，可以和任何数量的Source和Sink工作。
Sink	Sink负责将数据传输到下一跳或最终目的，成功完成后将数据从Channel移除。 典型的Sink类型如下： ● 存储数据到最终目的终端Sink，比如：HDFS、HBase。 ● 自动消耗的Sink，比如：Null Sink。 ● 用于Agent间通信的IPC Sink：Avro。 Sink必须作用于一个确切的Channel。

Flume也可以配置成多个Source、Channel、Sink，如图6-24所示：

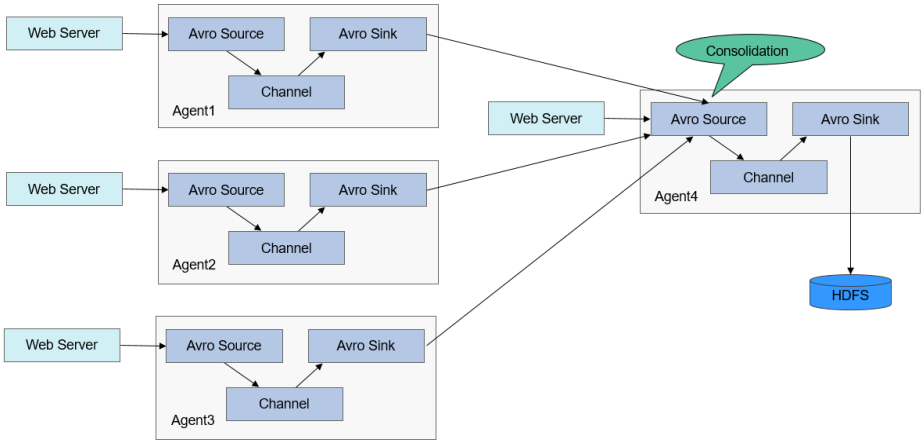
图 6-24 Flume 结构图



Flume的可靠性基于Agent间事务的交换，下一个Agent异常，Channel可以持久化数据，Agent恢复后再传输。Flume的可用性则基于内建的Load Balancing和Failover机制。Channel及Agent都可以配多个实体，实体之间可以使用负载分担等策略。每个Agent为一个JVM进程，同一台服务器可以有多个Agent。收集节点（Agent1，2，3）

负责处理日志，汇聚节点（Agent4）负责写入HDFS，每个收集节点的Agent可以选择多个汇聚节点，这样可以实现负载均衡。

图 6-25 Flume 级联结构图



Flume的架构和详细原理介绍，请参见：**https://flume.apache.org/releases/1.9.0.html**。

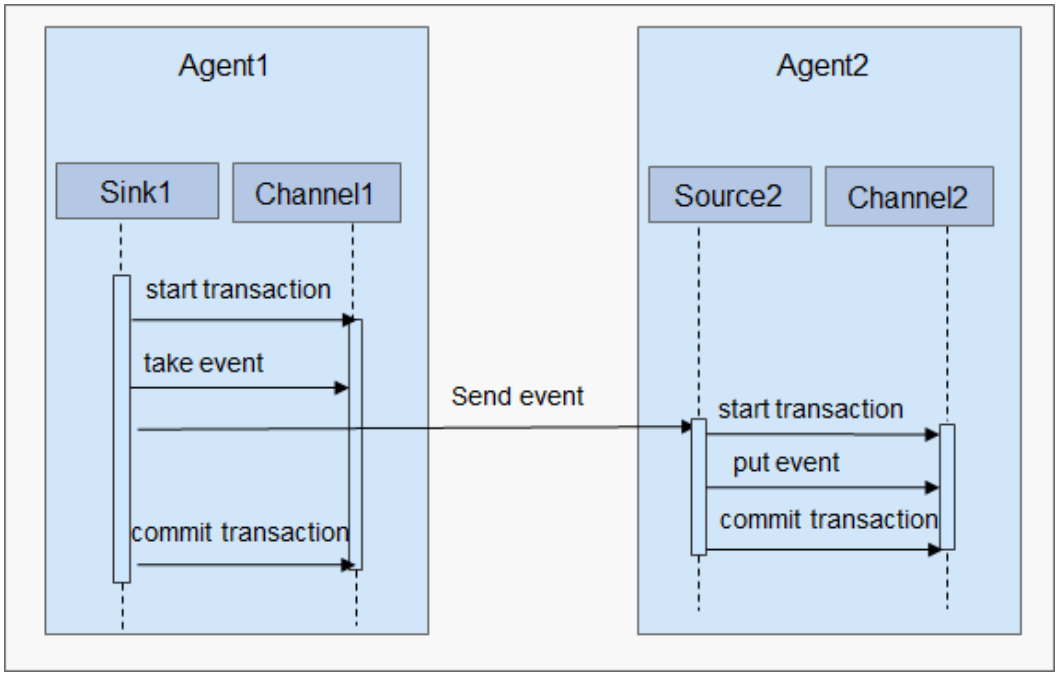
更多关于Flume组件操作指导，请参考**使用Flume**。

Flume 原理

Agent之间的可靠性

Agent之间数据交换流程如**图6-26**所示。

图 6-26 Agent 数据传输流程



1. Flume采用基于Transactions的方式保证数据传输的可靠性，当数据从一个Agent流向另外一个Agent时，两个Transactions已经开始生效。发送Agent的Sink首先从Channel取出一条消息，并且将该消息发送给另外一个Agent。如果接收消息的Agent成功地接收并处理消息，那么发送Agent将会提交Transactions，标识一次数据传输成功可靠地完成。
2. 当接收Agent接收到发送Agent发送的消息时，开始一个新的Transactions，当该数据被成功处理（写入Channel中），那么接收Agent提交该Transactions，并向发送Agent发送成功响应。
3. 如果在某次提交（commit）之前，数据传输出现了失败，将会再次开始上一次Transactions，并将上次发送失败的数据重新传输。因为commit操作已经将Transactions写入了磁盘，那么在进程故障退出并恢复业务之后，仍然可以继续上次的Transactions。

6.7.2 Flume 与其他组件的关系

Flume 与 HDFS 的关系

当用户配置HDFS作为Flume的Sink时，HDFS就作为Flume的最终数据存储系统，Flume将传输的数据全部按照配置写入HDFS中。

具体操作场景请参见**使用Flume服务端从本地采集静态日志保存到HDFS**和**使用Flume服务端从本地采集动态日志保存到HDFS**。

Flume 与 HBase 的关系

当用户配置HBase作为Flume的Sink时，HBase就作为Flume的最终数据存储系统，Flume将传输的数据全部按照配置写入HBase中。具体操作场景请参见**典型场景：从本地采集静态日志保存到HBase**。

6.7.3 Flume 开源增强特性

Flume 开源增强特性

- 提升传输速度。可以配置将指定的行数作为一个Event，而不仅是一行，提高了代码的执行效率以及减少写入磁盘的次数。
- 传输超大二进制文件。Flume根据当前内存情况，自动调整传输超大二进制文件的内存占用情况，不会导致Out of Memory（OOM）的出现。
- 支持定制传输前后准备工作。Flume支持定制脚本，指定在传输前或者传输后执行指定的脚本，用于执行准备工作。
- 管理客户端告警。Flume通过MonitorServer接收Flume客户端告警，并上报Manager告警管理中心。

6.8 Guardian

Guardian 基本原理

Guardian是一个在存算分离场景下为HDFS、Hive、Spark、Loader、HetuEngine等服务提供访问OBS的临时认证凭据的服务，只有对接OBS的场景下才需要安装Guardian组件。Guardian的典型特性包括：

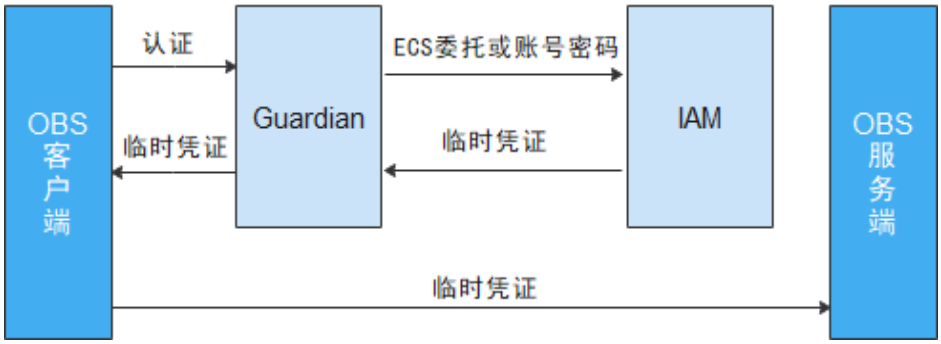
- 提供获取访问OBS的临时认证凭据的能力。
- 提供访问OBS的细粒度权限控制的能力。
- 提供访问OBS的临时认证凭据的统一缓存刷新能力。

Guardian服务端主要是TokenServer角色提供功能和能力，TokenServer支持多实例部署，每个实例都可以提供相同的功能，单点故障不影响服务功能，且对外提供RPC和HTTPS接口获取访问OBS的临时认证凭据。

Guardian 架构

Guardian的基本架构如图 Guardian架构所示。

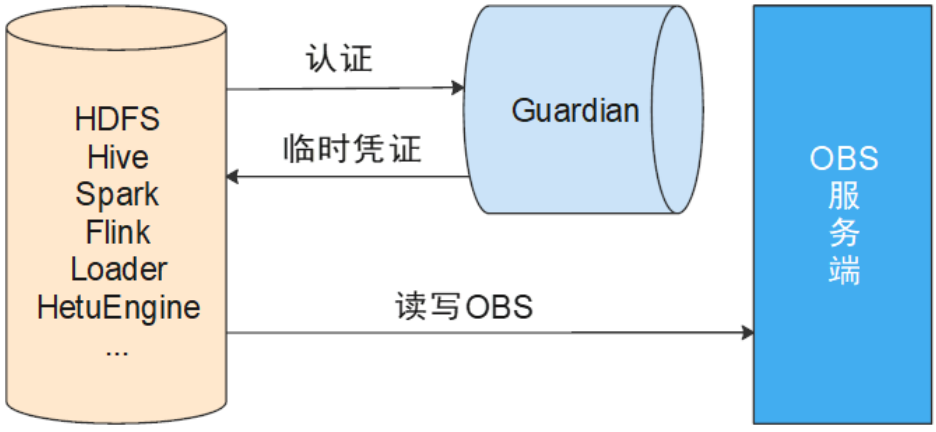
图 6-27 Guardian 架构



Guardian 与其他组件的关系

HDFS、Hive、Spark、Flink、Loader、HetuEngine在访问OBS之前，会先通过访问Guardian获取到访问OBS的临时凭据。Guardian会根据当前登录的用户去访问IAM请求生成带细粒度鉴权内容的临时凭据再返回给组件，组件拿此凭据去访问OBS，OBS根据凭据决定当前用户是否有权限访问。

图 6-28 Guardian 与其他组件的关系



6.9 HBase

6.9.1 HBase 基本原理

数据存储使用HBase来承接，HBase是一个开源的、面向列（Column-Oriented）、适合存储海量非结构化数据或半结构化数据的、具备高可靠性、高性能、可灵活扩展伸缩的、支持实时数据读写的分布式存储系统。更多关于HBase的信息，请参见：**https://hbase.apache.org/**。

存储在HBase中的表的典型特征：

- 大表（BigTable）：一个表可以有上亿行，上百万列。
- 面向列：面向列（族）的存储、检索与权限控制。
- 稀疏：表中为空（null）的列不占用存储空间。

MRS服务支持HBase组件的二级索引，支持为列值添加索引，提供使用原生的HBase接口的高性能基于列过滤查询的能力。

更多关于HBase组件操作指导，请参考**使用HBase**。

HBase 结构

HBase集群由主备Master进程和多个RegionServer进程组成。如**图6-29**所示。

图 6-29 HBase 结构

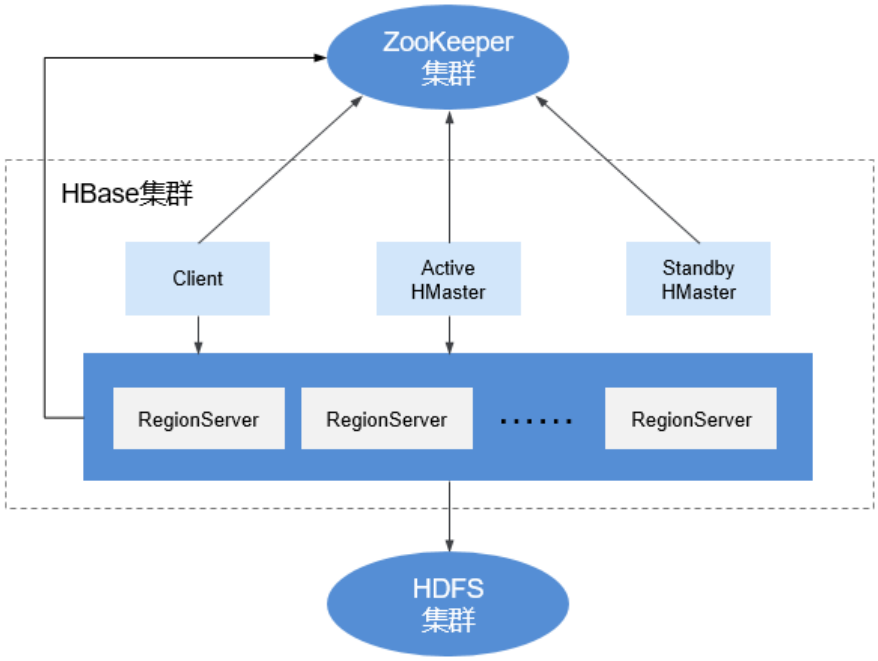


表 6-6 模块说明

名称	描述
Master	又叫HMaster，在HA模式下，包含主用Master和备用Master。 ● 主用Master：负责HBase中RegionServer的管理，包括表的增、删、改、查；RegionServer的负载均衡，Region分布调整；Region分裂以及分裂后的Region分配；RegionServer失效后的Region迁移等。 ● 备用Master：当主用Master故障时，备用Master将取代主用Master对外提供服务。故障恢复后，原主用Master降为备用。
Client	Client使用HBase的RPC机制与Master、RegionServer进行通信。Client与Master进行管理类通信，与RegionServer进行数据操作类通信。
RegionServer	RegionServer负责提供表数据读写等服务，是HBase的数据处理和计算单元。 RegionServer一般与HDFS集群的DataNode部署在一起，实现数据的存储功能。
ZooKeeper集群	ZooKeeper为HBase集群中各进程提供分布式协作服务。各RegionServer将信息注册到ZooKeeper中，主用Master据此感知各个RegionServer的健康状态。
HDFS集群	HDFS为HBase提供高可靠的文件存储服务，HBase的数据全部存储在HDFS中。

HBase 原理

- **HBase数据模型**

  HBase以表的形式存储数据，数据模型如**图 HBase数据模型**所示。表中的数据划分为多个Region，并由Master分配给对应的RegionServer进行管理。

  每个Region包含了表中一段RowKey区间范围内的数据，HBase的一张数据表开始只包含一个Region，随着表中数据的增多，当一个Region的大小达到容量上限后会分裂成两个Region。可以在创建表时定义Region的RowKey区间，或者在配置文件中定义Region的大小。

图 6-30 HBase 数据模型

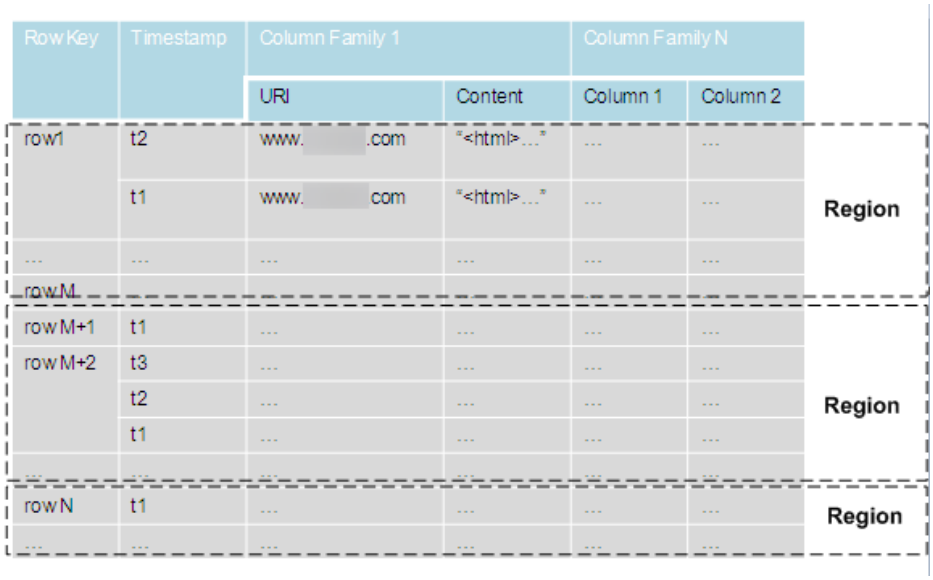


表 6-7 概念介绍

名称	描述
RowKey	行键，相当于关系表的主键，每一行数据的唯一标识。字符串、整数、二进制串都可以作为RowKey。所有记录按照RowKey排序后存储。
Timestamp	每次数据操作对应的时间戳，数据按时间戳区分版本，每个Cell的多个版本的数据按时间倒序存储。
Cell	HBase最小的存储单元，由Key和Value组成。Key由row、column family、column qualifier、timestamp、type、MVCC version这6个字段组成。Value就是对应存储的二进制数据对象。
Column Family	列族，一个表在水平方向上由一个或多个Column Family组成。一个CF（Column Family）可以由任意多个Column组成。Column是CF下的一个标签，可以在写入数据时任意添加，因此CF支持动态扩展，无需预先定义Column的数量和类型。HBase中表的列非常稀疏，不同行的列的个数和类型都可以不同。此外，每个CF都有独立的生存周期（TTL）。可以只对行上锁，对行的操作始终是原始的。
Column	与传统的数据库类似，HBase的表中也有列的概念，列用于表示相同类型的数据。

- **RegionServer数据存储**

  RegionServer主要负责管理由HMaster分配的Region，RegionServer的数据存储结构如**图 RegionServer的数据存储结构**所示。

图 6-31 RegionServer 的数据存储结构

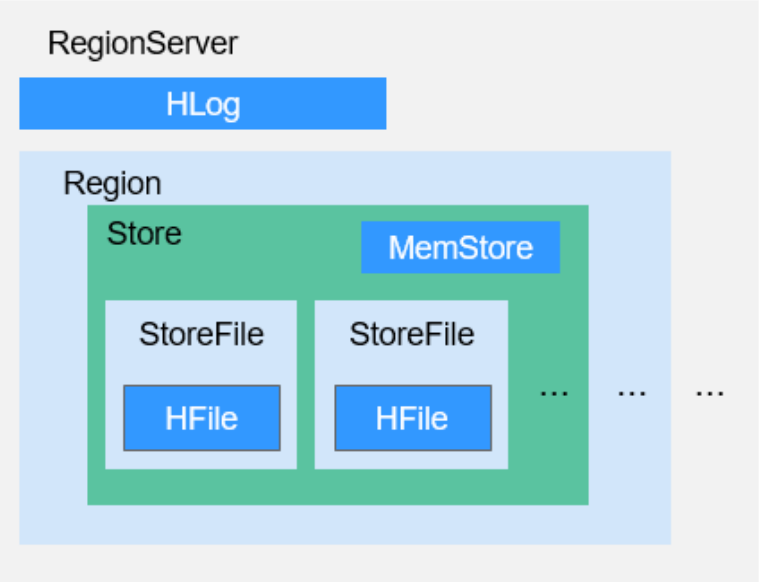


图 RegionServer的数据存储结构中Region的各部分的说明如**表 Region结构说明**所示。

表 6-8 Region 结构说明

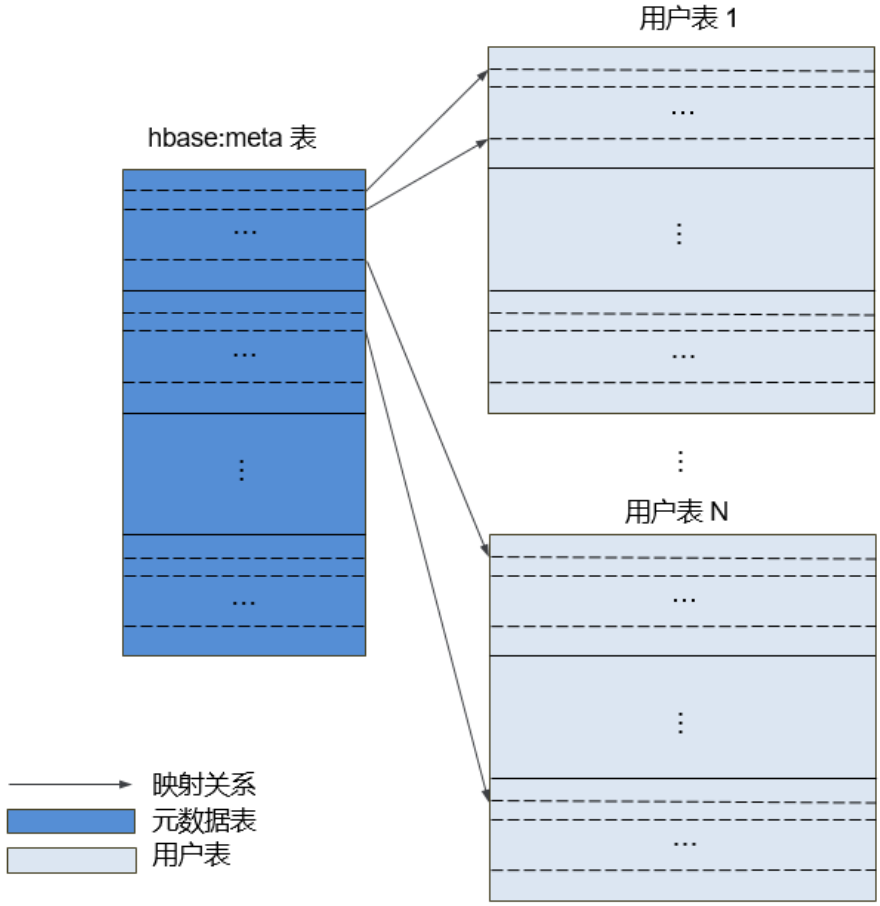
名称	描述
Store	一个Region由一个或多个Store组成，每个Store对应**图 HBase数据模型**中的一个Column Family。
MemStore	一个Store包含一个MemStore，MemStore缓存客户端向Region插入的数据，当RegionServer中的MemStore大小达到配置的容量上限时，RegionServer会将MemStore中的数据flush到HDFS中。
StoreFile	MemStore的数据flush到HDFS后成为StoreFile，随着数据的插入，一个Store会产生多个StoreFile，当StoreFile的个数达到配置的最大值时，RegionServer会将多个StoreFile合并为一个大的StoreFile。
HFile	HFile定义了StoreFile在文件系统中的存储格式，它是当前HBase系统中StoreFile的具体实现。
HLog	HLog日志保证了当RegionServer故障的情况下用户写入的数据不丢失，RegionServer的多个Region共享一个相同的HLog。

- **元数据表**

  元数据表是HBase中一种特殊的表，用来帮助Client定位到具体的Region。元数据表包括“hbase:meta”表，用于记录用户表的Region信息，例如，Region位置、起始RowKey及结束RowKey等信息。

  元数据表和用户表的映射关系如**图 元数据表和用户表的映射关系**所示。

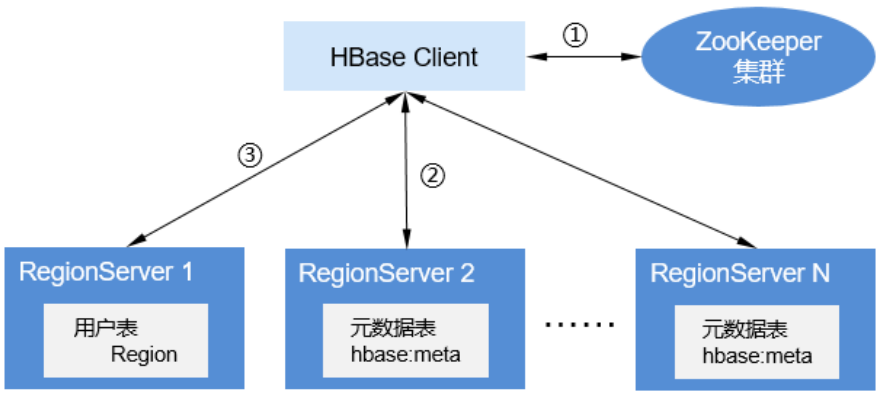
图 6-32 元数据表和用户表的映射关系



- **数据操作流程**

  HBase数据操作流程如**图 数据操作流程**所示。

图 6-33 数据操作流程



a. 对HBase进行增、删、改、查数据操作时，HBase Client首先连接ZooKeeper获得“hbase:meta”表所在的RegionServer的信息（涉及NameSpace级别修改的，比如创建表、删除表需要访问HMaster更新meta信息）。

b. HBase Client连接到包含对应的“hbase:meta”表的Region所在的RegionServer，并获得相应的用户表的Region所在的RegionServer位置信息。

c. HBase Client连接到对应的用户表Region所在的RegionServer，并将数据操作命令发送给该RegionServer，RegionServer接收并执行该命令从而完成本次数据操作。

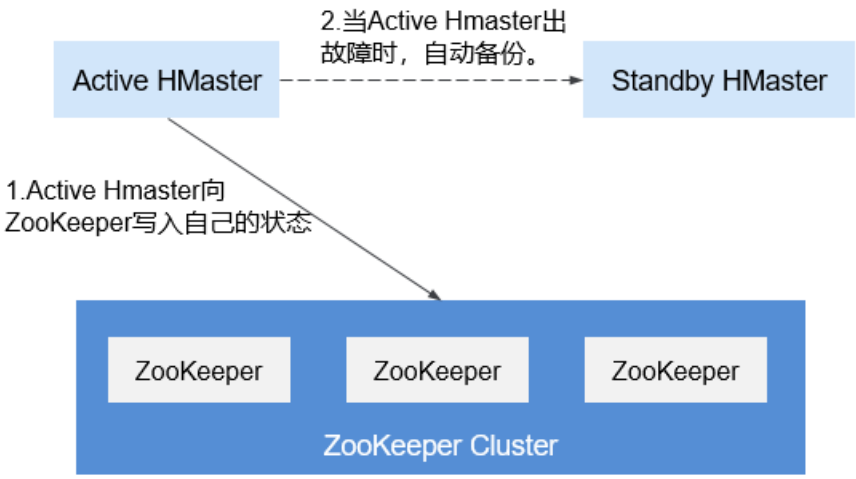
为了提升数据操作的效率，HBase Client会在内存中缓存“hbase:meta”和用户表Region的信息，当应用程序发起下一次数据操作时，HBase Client会首先从内存中获取这些信息；当未在内存缓存中找到对应数据信息时，HBase Client会重复上述操作。

6.9.2 HBase HA 方案介绍

HBase HA 原理与实现方案

HBase中的HMaster负责Region分配，当RegionServer服务停止后，HMaster会把相应Region迁移到其他RegionServer。为了解决HMaster单点故障导致HBase正常功能受到影响的问题，引入HMaster HA模式。

图 6-34 HMaster 高可用性实现架构



HMaster高可用性架构是通过在ZooKeeper集群创建Ephemeral node（临时节点）实现的。

当HMaster两个节点启动时都会尝试在ZooKeeper集群上创建一个znode节点Master，先创建的成为Active HMaster，后创建的成为Standby HMaster。

Standby HMaster会在Master节点添加监测事件。如果主节点服务停止，就会和ZooKeeper集群失去联系，session过期之后Master节点会消失。Standby节点通过监测事件（watch event）感知到节点消失，会去创建Master节点自己成为Active HMaster，主备倒换完成。如果后续停止服务的节点重新启动，发现Master节点已经存在，则进入Standby模式，并对Master znode创建监测事件。

当客户端访问HBase时，会首先通过ZooKeeper上的Master节点信息找到HMaster的地址，然后与Active HMaster进行连接。

6.9.3 HBase 与其他组件的关系

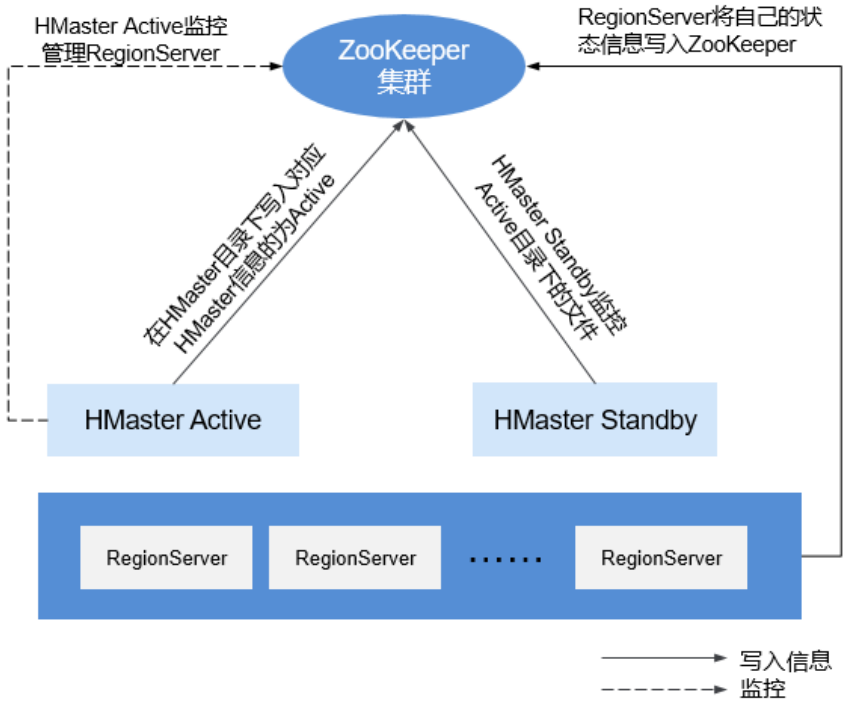
HBase 和 HDFS 的关系

HDFS是Apache的Hadoop项目的子项目，HBase利用Hadoop HDFS作为其文件存储系统。HBase位于结构化存储层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持。除了HBase产生的一些日志文件，HBase中的所有数据文件都可以存储在Hadoop HDFS文件系统上。

HBase 和 ZooKeeper 的关系

HBase和ZooKeeper的关系如图 ZooKeeper和HBase的关系所示。

图 6-35 HBase 和 ZooKeeper 的关系



1. HRegionServer以Ephemeral node的方式注册到ZooKeeper中。其中ZooKeeper存储HBase的如下信息：HBase元数据、HMaster地址。
2. HMaster通过ZooKeeper随时感知各个HRegionServer的健康状况，以便进行控制管理。
3. HBase也可以部署多个HMaster，类似HDFS NameNode，当HMaster主节点出现故障时，HMaster备用节点会通过ZooKeeper获取主HMaster存储的整个HBase集群状态信息。即通过ZooKeeper实现避免HBase单点故障问题的问题。

6.9.4 HBase 开源增强特性

HBase 开源增强特性：HIndex

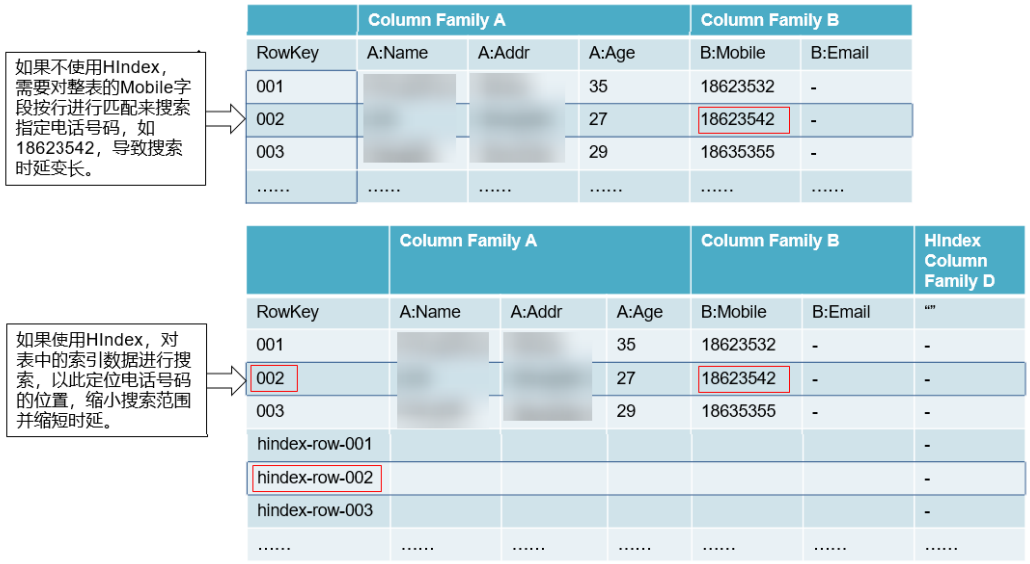
HBase是一个Key-Value类型的分布式存储数据库。每张表的数据按照RowKey的字典顺序排序，因此，如果按照某个指定的RowKey去查询数据，或者指定某一个RowKey

范围去扫描数据时，HBase可以快速定位到需要读取的数据位置，从而可以高效地获取到所需要的数据。

在实际应用中，很多场景是查询某一个列值为“XXX”的数据。HBase提供了Filter特性去支持这样的查询，它的原理是：按照RowKey的顺序，去遍历所有可能的数据，再依次去匹配那一列的值，直到获取到所需要的数据。可以看出，可能只是为了获取一行数据，它却扫描了很多不必要的数据。因此，如果对于这样的查询请求非常频繁并且对查询性能要求较高，使用Filter无法满足这个需求。

这就是HBase HIndex产生的背景。HIndex为HBase提供了按照某些列的值进行索引的能力。

图 6-36 HIndex



- 索引数据不支持滚动升级。
- 组合索引限制。
  - 用户必须在单次mutation中输入或删除参与组合索引的所有列。否则会导致不一致问题。

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

    正确的写操作：

    ```
    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);
    ```

    错误的写操作：

    ```
    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);
    ```

  - 使用组合条件查询，仅支持组合索引列包含过滤条件的查询，或者不指定StartRow和StopRow的部分索引列的查询。

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

正确的查询操作：

```
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)",STARTROW=>'row001',STOPROW
=>'row100'}
```

错误的查询操作：

```
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)" ,STARTROW=>'row001',STOPROW
=>'row100' }
```

- 用户不能为有索引数据的表配置任何分裂策略。
- 不支持其他的mutation操作，如increment和append。
- 不支持maxVersions>1的列的索引。
- 不支持一行数据索引列的更新操作。

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

  索引2：IDX2=>cf2:[q2->datatype]

  正确的更新操作：

```
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);
```

  错误的更新操作：

```
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);
```

- 添加索引的表不应拥有大于32KB的值。
- 当由于列族级TTL（生存周期）过期而导致用户数据删除时，对应的索引数据不会立即删除。索引数据会在进行major compaction操作时被删除。
- 用户列族的TTL在索引创建后不能修改。
  - 如果在创建索引之后，列族的TTL值变大，应该删除并重新创建该索引。否则，一些已经生成的索引数据会先于用户数据被删除。
  - 如果在创建索引之后，列族的TTL值变小。索引数据会晚于用户数据被删除。
- 索引查询不支持reverse；且查询结果是无序的。
- 索引不支持clone snapshot操作。
- 索引表必须使用HIndexWALPlayer回放日志，不支持WALPlayer回放日志。

```
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
```

- 使用deleteall操作索引表存在性能慢问题。
- 索引表不支持HBCK。如需使用HBCK修复索引表，需先删除索引数据后，再进行修复。

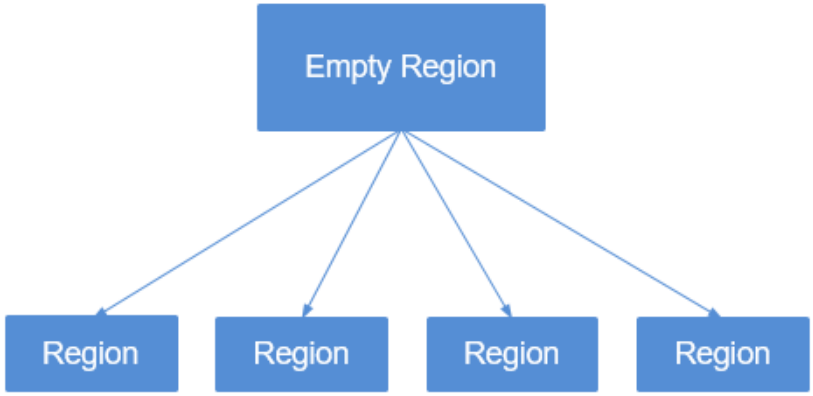
HBase 开源增强特性：支持多点分割

当用户在HBase创建Region预先分割的表时，用户可能不知道数据的分布趋势，所以Region的分割可能不合适，所以当系统运行一段时间后，Region需要重新分割以获得更好的查询性能，HBase只会分割空的Region。

HBase自带的Region分割只有当Region到达设定的Threshold后才会进行分割，这种分割被称为单点分割。

为了实现根据用户的需要动态分割Region以获得更好的性能这一目标，开发了多点分割又称动态分割，即把空的Region预先分割成多个Region。通过预先分割，避免了因为Region空间不足出现Region分割导致性能下降的现象。

图 6-37 多点分割



HBase 开源增强特性：连接数限制

过多的session连接意味着过多的查询和MR任务跑在HBase上，这会导致HBase性能下降以至于导致HBase拒绝服务。通过配置参数来限制客户端连接到HBase服务器端的session数目，来实现HBase过载保护。

HBase 开源增强特性：容灾增强

主备集群之间的容灾能力可以增强HBase数据的高可用性，主集群提供数据服务，备用集群提供数据备份，当主集群出现故障时，备集群可以提供数据服务。相比开源Replication功能，做了如下增强：

1. 备集群白名单功能，只接受指定集群IP的数据推送。
2. 开源版本中replication是基于WAL同步，在备集群回放WAL实现数据备份的。对于BulkLoad，由于没有WAL产生，BulkLoad的数据不会replicate到备集群。通过将BulkLoad操作记录在WAL上，同步至备集群，备集群通过WAL读取BulkLoad操作记录，将对应的主集群的HFile加载到备集群，完成数据的备份。
3. 开源版本中HBase对于系统表ACL做了过滤，ACL信息不会同步至备集群，通过新加一个过滤器 *org.apache.hadoop.hbase.replication.SystemTableWALEntryFilterAllowACL*，允许ACL信息同步至备集群，用户可以通过配置 *hbase.replication.filter.sytemWALEntryFilter*使用该过滤器实现ACL同步。
4. 备集群只读限制，备集群只接受备集群节点内的内置管理用户对备集群的HBase进行修改操作，即备集群节点之外的HBase客户端只能对备集群的HBase进行读操作。

HBase 开源增强特性：HFS

HBase文件存储模块（HBase FileStream，简称HFS）是HBase的独立模块，它作为对HBase与HDFS接口的封装，应用在MRS的上层应用，为上层应用提供文件的存储、读取、删除等功能。

在Hadoop生态系统中，无论是HDFS，还是HBase，均在面对海量文件的存储的时候，在某些场景下，都会存在一些很难解决的问题：

- 如果把海量小文件直接保存在HDFS中，会给NameNode带来极大的压力。

- 由于HBase接口以及内部机制的原因，一些较大的文件也不适合直接保存到HBase中。

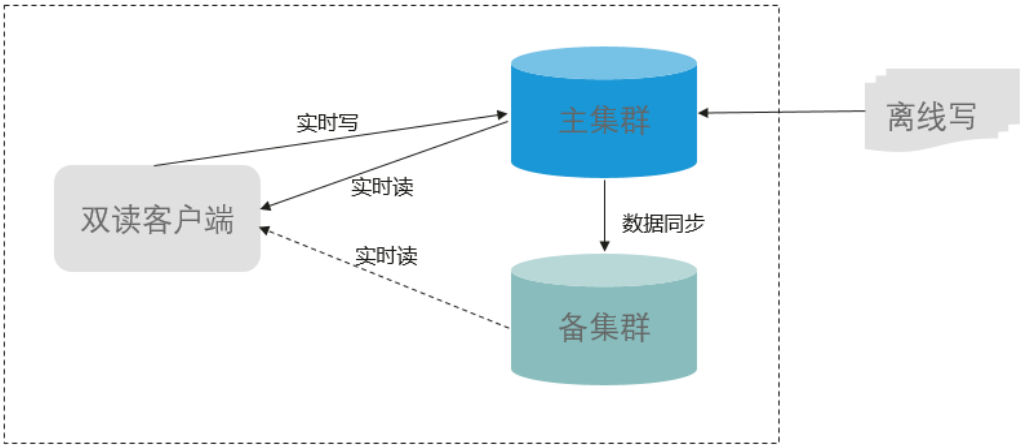
HFS的出现，就是为了解决需要在Hadoop中存储海量小文件，同时也要存储一些大文件的混合的场景。简单来说，就是在HBase表中，需要存放大量的小文件（10MB以下），同时又需要存放一些比较大的文件（10MB以上）。

HFS为以上场景提供了统一的操作接口，这些操作接口与HBase的函数接口类似。

HBase 开源增强特性：HBase 双读

在HBase存储场景下，因为GC、网络抖动、磁盘坏道等原因，很难保证99.9%的查询稳定性。为了满足用户大数据量随机读低毛刺的要求，新增了HBase双读特性。

HBase双读特性是建立在主备集群容灾能力之上，两套集群同时产生毛刺的概率要远远小于一套集群，即采用双集群并发访问的方式，保证查询的稳定性。当用户发起查询请求时，同时查询两个集群的HBase服务，在等待一段时间（最大容忍的毛刺时间）后，如果主集群没有返回结果，则可以使用响应最快的集群数据。原理图如下：



HBase 开源增强特性：Phoenix CsvBulkLoad 工具导入支持用户自定义分隔符

说明

该内容适用于MRS 3.2.0及之后版本。

Phoenix开源CsvBulkLoad工具当前仅支持指定单个字符作为数据分隔符，当用户数据文件中可能包含任意字符时，一般会采用特殊的字符串作为分隔符，为了满足此类场景，增加了对用户自定义分隔符的支持，用户可以采用限定长度内的任意可见字符进行组合作为分隔符来导入数据文件。

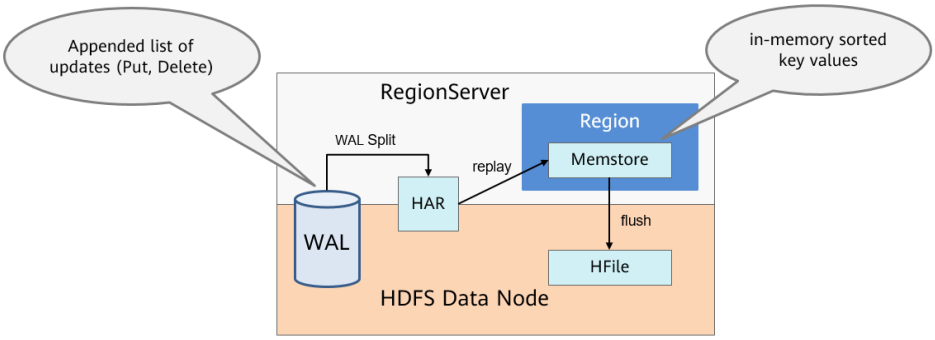
HBase 开源增强特性：使用 HAR 文件格式拆分 WAL 文件

说明

该内容适用于MRS 3.2.0及之后版本。

当RegionServer发生故障或重启时，HMaster会使用ServerCrashProcedure对RegionServer的业务进行恢复，恢复过程中包括拆分WAL文件。在WAL文件拆分过程中，会产生大量的小文件，可能造成HDFS的性能瓶颈，导致服务恢复时间过长。

本功能主要在拆分过程中将原本的小文件写入到HAR文件中，旨在减少拆分WAL过程中产生的小文件，从而缩短RegionServer恢复时长。



HBase 开源增强特性：Batch TRSP

HBase 2.x内核版本使用HBase Procedure框架重写了region assignment的逻辑（AMV2）。每个Region的open或者close都会有一个TransitRegionStateProcedure(TRSP)与之关联。当RegionServer因为故障或重启需要恢复业务时，HMaster会为每个需要恢复的Region创建一个TRSP，大量的TRSP需要把数据持久化到Proc WAL文件中并且需要跟RegionServer进行RPC交互，可能造成HMaster性能瓶颈，导致服务恢复时间过长。

本功能主要通过在TRSP中添加attach region的方式，利用一个TRSP将一个RegionServer所有的Region进行恢复处理，RegionServer也将进行Region的批量open/close并一次性全部上报给HMaster。

说明

该特性只支持将Region恢复到原来的RegionServer，因此该优化生效的前提为HMaster在创建TRSP时，故障或重启的RegionServer已经重新上线。因此，该特性主要用于优化HBase重启或者服务故障恢复的时长，如果是少量RegionServer发生故障，可能因为HMaster在RegionServer重新上线前已经创建了TRSP而不生效。

该内容适用于MRS 3.2.0及之后版本。

6.10 HDFS

6.10.1 HDFS 基本原理

HDFS是Hadoop的分布式文件系统（Hadoop Distributed File System），实现大规模数据可靠的分布式读写。HDFS针对的使用场景是数据读写具有“一次写，多次读”的特征，而数据“写”操作是顺序写，也就是在文件创建时的写入或者在现有文件之后的添加操作。HDFS保证一个文件在一个时刻只被一个调用者执行写操作，而可以被多个调用者执行读操作。

更多关于HDFS组件操作指导，请参考**使用HDFS**。

说明

如需使用HDFS，请确保MRS集群内已安装Hadoop服务。

HDFS 结构

HDFS包含主、备NameNode和多个DataNode，如图6-38所示。

HDFS是一个Master/Slave的架构，在Master上运行NameNode，而在每一个Slave上运行DataNode，ZKFC需要和NameNode一起运行。

NameNode和DataNode之间的通信都是建立在TCP/IP的基础之上的。NameNode、DataNode、ZKFC和JournalNode能部署在运行Linux的服务器上。

图 6-38 HA HDFS 结构

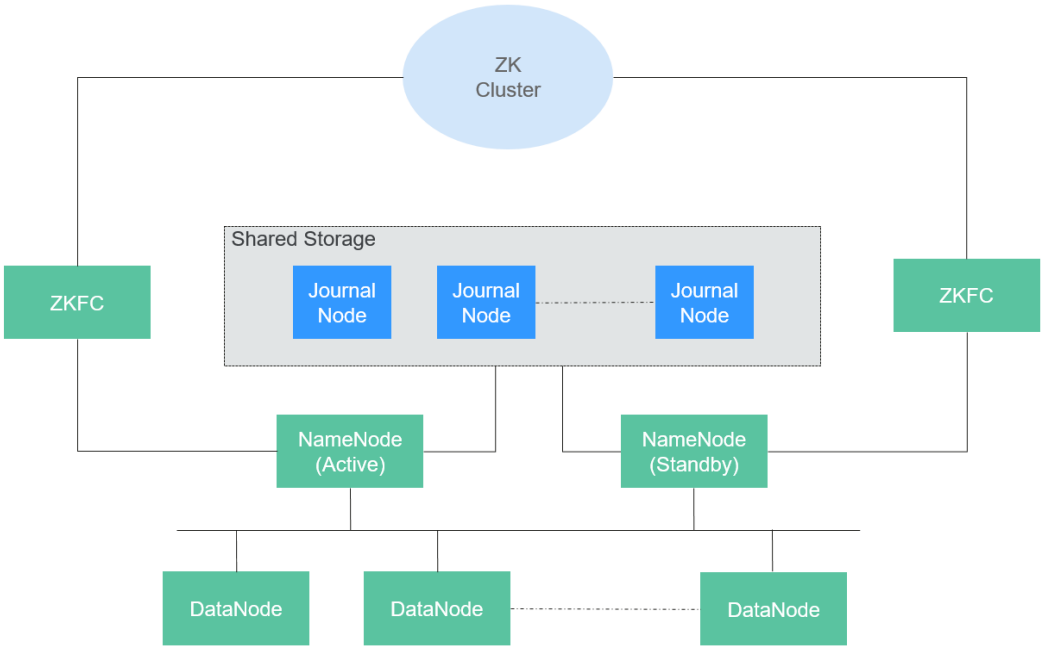


图6-38中各模块的功能说明如表6-9所示。

表 6-9 模块说明

名称	描述
NameNode	用于管理文件系统的命名空间、目录结构、元数据信息以及提供备份机制等，分为： • Active NameNode：管理文件系统的命名空间、维护文件系统的目录结构树以及元数据信息；记录写入的每个“数据块”与其归属文件的对应关系。 • Standby NameNode：与Active NameNode中的数据保持同步；随时准备在Active NameNode出现异常时接管其服务。 • Observer NameNode：与Active NameNode中的数据保持同步，处理来自客户端的读请求。
DataNode	用于存储每个文件的“数据块”数据，并且会周期性地向NameNode报告该DataNode的数据存放情况。
JournalNode	HA集群下，用于同步主备NameNode之间的元数据信息。

名称	描述
ZKFC	ZKFC是需要和NameNode一一对应的服务，即每个NameNode都需要部署ZKFC。它负责监控NameNode的状态，并及时把状态写入ZooKeeper。ZKFC也有选择谁作为Active NameNode的权利。
ZK Cluster	ZooKeeper是一个协调服务，帮助ZKFC执行主NameNode的选举。
HttpFS gateway	HttpFS是个单独无状态的gateway进程，对外提供webHDFS接口，对HDFS使用FileSystem接口对接。可用于不同Hadoop版本间的数据传输，及用于访问在防火墙后的HDFS（HttpFS用作gateway）。

- **HDFS HA架构**

  HA即为High Availability，用于解决NameNode单点故障问题，该特性通过主备的方式为主NameNode提供一个备用者，一旦主NameNode出现故障，可以迅速切换至备NameNode，从而不间断对外提供服务。

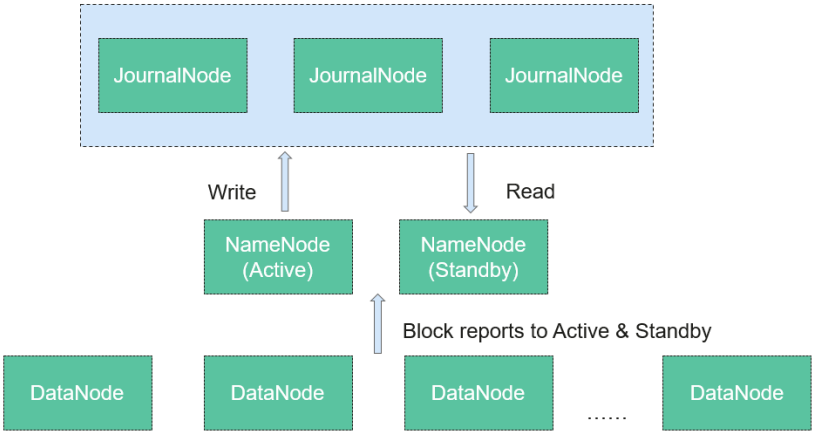
  在一个典型HDFS HA场景中，通常由两个NameNode组成，一个处于Active状态，另一个处于Standby状态。

  为了能实现Active和Standby两个NameNode的元数据信息同步，需提供一个共享存储系统。本版本提供基于QJM（Quorum Journal Manager）的HA解决方案，如图6-39所示。主备NameNode之间通过一组JournalNode同步元数据信息。

  通常配置奇数个（2N+1个）JournalNode，且最少要运行3个JournalNode。这样，一条元数据更新消息只要有N+1个JournalNode写入成功就认为数据写入成功，此时最多容忍N个JournalNode写入失败。比如，3个JournalNode时，最多允许1个JournalNode写入失败，5个JournalNode时，最多允许2个JournalNode写入失败。

  由于JournalNode是一个轻量级的守护进程，可以与Hadoop其它服务共用机器。建议将JournalNode部署在控制节点上，以避免数据节点在进行大数据量传输时引起JournalNode写入失败。

  图 6-39 基于 QJM 的 HDFS 架构

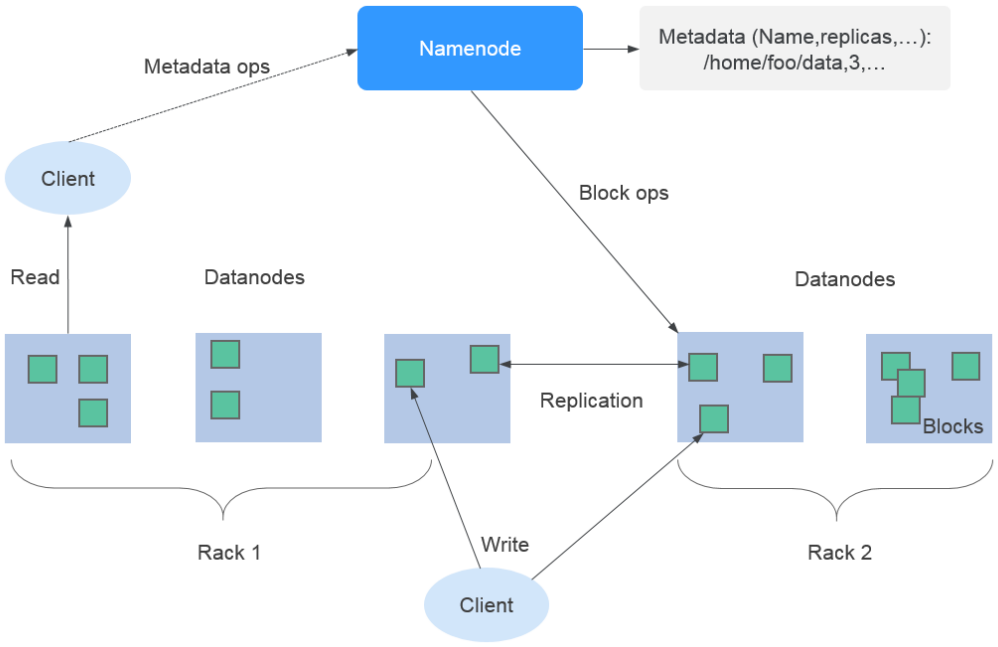


HDFS 可靠性

MRS使用HDFS的副本机制来保证数据的可靠性，HDFS中每保存一个文件则自动生成1个备份文件，即共2个副本。HDFS副本数可通过“dfs.replication”参数查询。

- 当MRS集群中Core节点规格选择为非本地盘（hdd）时，若集群中只有一个Core节点，则HDFS默认副本数为1。若集群中Core节点数大于等于2，则HDFS默认副本数为2。
- 当MRS集群中Core节点规格选择为本地盘（hdd）时，若集群中只有一个Core节点，则HDFS默认副本数为1。若集群中有两个Core节点，则HDFS默认副本数为2。若集群中Core节点数大于等于3，则HDFS默认副本数为3。

图 6-40 HDFS 架构



MRS支持HDFS组件上节点均衡调度和单节点内的磁盘均衡调度，有助于扩容节点或扩容磁盘后的HDFS存储性能提升。

6.10.2 HDFS HA 方案介绍

HDFS HA 方案背景

在Hadoop 2.0.0之前，HDFS集群中存在单点故障问题。由于每个集群只有一个NameNode，如果NameNode所在机器发生故障，将导致HDFS集群无法使用，除非NameNode重启或者在另一台机器上启动。这在两个方面影响了HDFS的整体可用性：

1. 当异常情况发生时，如机器崩溃，集群将不可用，除非重新启动NameNode。
2. 计划性的维护工作，如软硬件升级等，将导致集群停止工作。

针对以上问题，HDFS高可用性方案通过自动或手动（可配置）的方式，在一个集群中为NameNode启动一个热替换的NameNode备份。当一台机器故障时，可以迅速地自动进行NameNode主备切换。或者当主NameNode节点需要进行维护时，通过MRS集

群管理员控制，可以手动进行NameNode主备切换，从而保证集群在维护期间的可用性。

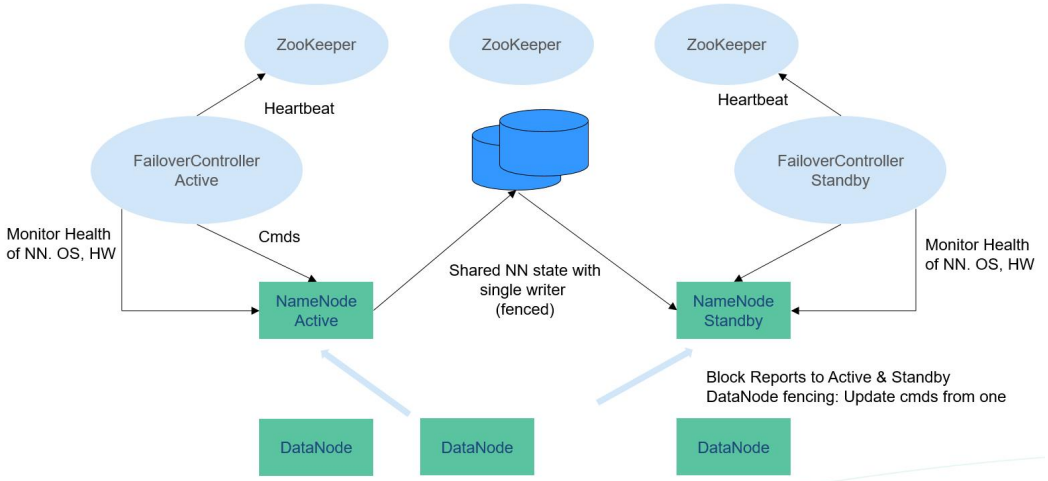
有关HDFS自动故障转移功能，请参阅：

MRS 3.2.0之前版本：**http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html#Automatic_Failover**

MRS 3.2.0及之后版本：**https://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html#Automatic_Failover**

HDFS HA 实现方案

图 6-41 典型的 HA 部署方式



在一个典型的HA集群中（如图6-41），需要把两个NameNodes配置在两台独立的机器上。在任何一个时间点，只有一个NameNode处于Active状态，另一个处于Standby状态。Active节点负责处理所有客户端操作，Standby节点时刻保持与Active节点同步的状态以便在必要时进行快速主备切换。

为保持Active和Standby节点的数据一致性，两个节点都要与一组称为JournalNode的节点通信。当Active对文件系统元数据进行修改时，会将其修改日志保存到大多数的JournalNode节点中，例如有3个JournalNode，则日志会保存在至少2个节点中。Standby节点监控JournalNodes的变化，并同步来自Active节点的修改。根据修改日志，Standby节点将变动应用到本地文件系统元数据中。一旦发生故障转移，Standby节点能够确保与Active节点的状态是一致的。这保证了文件系统元数据在故障转移时在Active和Standby之间是完全同步的。

为保证故障转移快速进行，Standby需要时刻保持最新的块信息，为此DataNodes同时向两个NameNodes发送块信息和心跳。

对一个HA集群，保证任何时刻只有一个NameNode是Active状态至关重要。否则，命名空间会分为两部分，有数据丢失和产生其他错误的风险。为保证这个属性，防止“split-brain”问题的产生，JournalNodes在任何时刻都只允许一个NameNode写入。在故障转移时，将变为Active状态的NameNode获得写入JournalNodes的权限，这会有效防止其他NameNode的Active状态，使得切换安全进行。

关于HDFS高可用性方案的更多信息，可参考如下链接：

MRS 3.2.0之前版本：**http://hadoop.apache.org/docs/r3.1.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html**

MRS 3.2.0及之后版本：**https://hadoop.apache.org/docs/r3.3.1/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html**

6.10.3 HDFS 与其他组件的关系

HDFS 和 HBase 的关系

HDFS是Apache的Hadoop项目的子项目，HBase利用Hadoop HDFS作为其文件存储系统。HBase位于结构化存储层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持。除了HBase产生的一些日志文件，HBase中的所有数据文件都可以存储在Hadoop HDFS文件系统上。

HDFS 和 MapReduce 的关系

- HDFS是Hadoop分布式文件系统，具有高容错和高吞吐量的特性，可以部署在价格低廉的硬件上，存储应用程序的数据，适合有超大数据集的应用程序。
- 而MapReduce是一种编程模型，用于大数据集（大于1TB）的并行运算。在MapReduce程序中计算的数据可以来自多个数据源，如Local FileSystem、HDFS、数据库等。最常用的是HDFS，可以利用HDFS的高吞吐性能读取大规模的数据进行计算。同时在计算完成后，也可以将数据存储到HDFS。

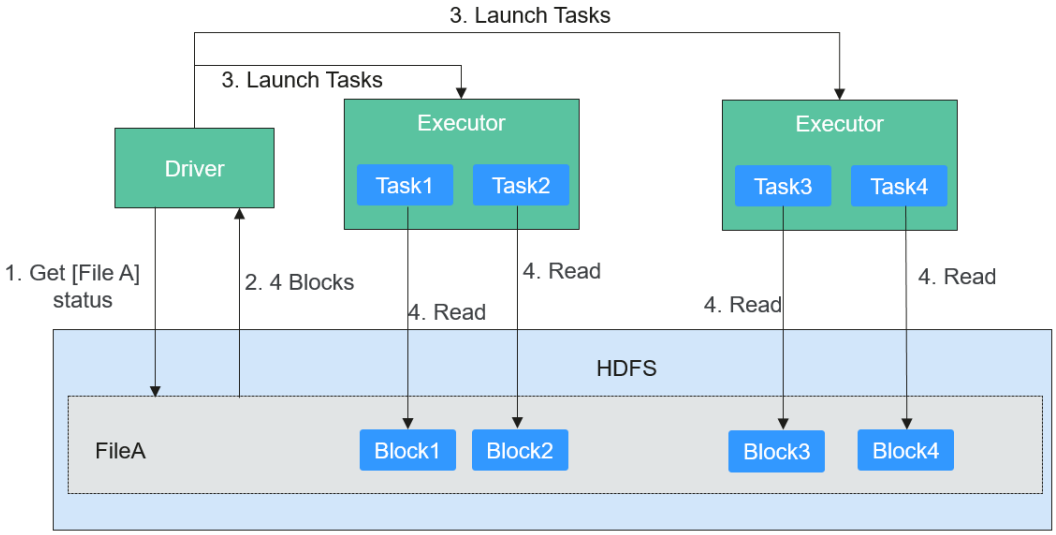
HDFS 和 Spark 的关系

通常，Spark中计算的数据可以来自多个数据源，如Local File、HDFS等。最常用的是HDFS，用户可以一次读取大规模的数据进行并行计算。在计算完成后，也可以将数据存储到HDFS。

分解来看，Spark分成控制端（Driver）和执行端（Executor）。控制端负责任务调度，执行端负责任务执行。

读取文件的过程如**图6-42**所示。

图 6-42 读取文件过程

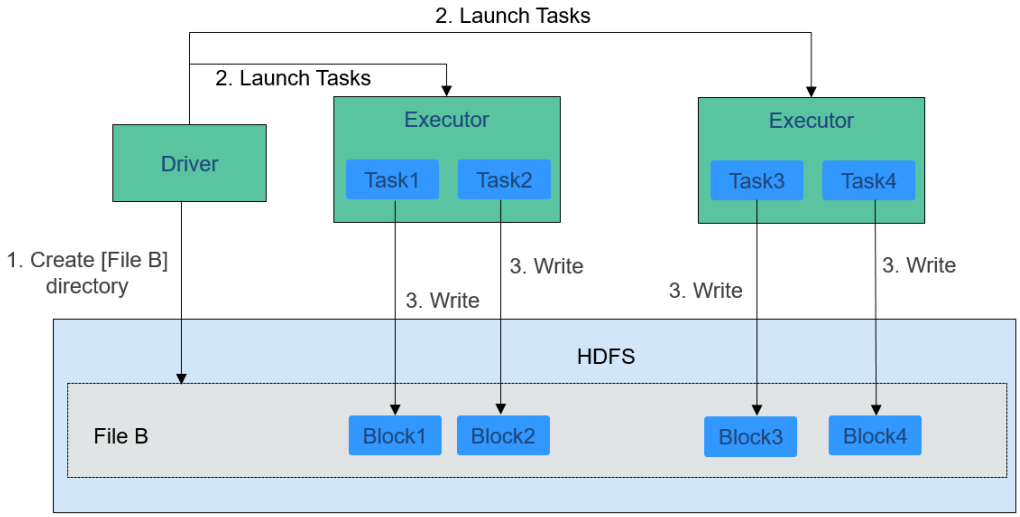


读取文件步骤的详细描述如下所示：

1. Driver与HDFS交互获取File A的文件信息。
2. HDFS返回该文件具体的Block信息。
3. Driver根据具体的Block数据量，决定一个并行度，创建多个Task去读取这些文件Block。
4. 在Executor端执行Task并读取具体的Block，作为RDD（弹性分布数据集）的一部分。

写入文件的过程如图6-43所示。

图 6-43 写入文件过程



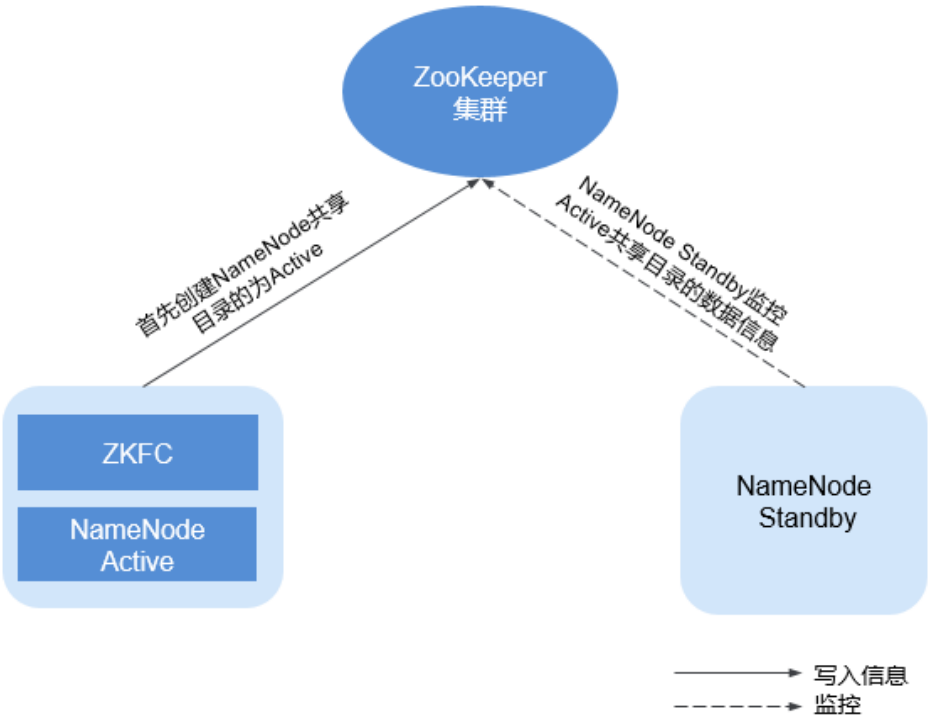
HDFS文件写入的详细步骤如下所示：

1. Driver创建要写入文件的目录。
2. 根据RDD分区分块情况，计算出写数据的Task数，并下发这些任务到Executor。
3. Executor执行这些Task，将具体RDD的数据写入到步骤1创建的目录下。

HDFS 和 ZooKeeper 的关系

ZooKeeper与HDFS的关系如图6-44所示。

图 6-44 ZooKeeper 和 HDFS 的关系



ZKFC（ZKFailoverController）作为一个ZooKeeper集群的客户端，用来监控NameNode的状态信息。ZKFC进程仅在部署了NameNode的节点中存在。HDFS NameNode的Active和Standby节点均部署有zkfc进程。

1. HDFS NameNode的ZKFC连接到ZooKeeper，把主机名等信息保存到ZooKeeper中，即“/hadoop-ha”下的znode目录里。先创建znode目录的NameNode节点为主节点，另一个为备节点。HDFS NameNode Standby通过ZooKeeper定时读取NameNode信息。
2. 当主节点进程异常结束时，HDFS NameNode Standby通过ZooKeeper感知“/hadoop-ha”目录下发生了变化，NameNode会进行主备切换。

6.10.4 HDFS 开源增强特性

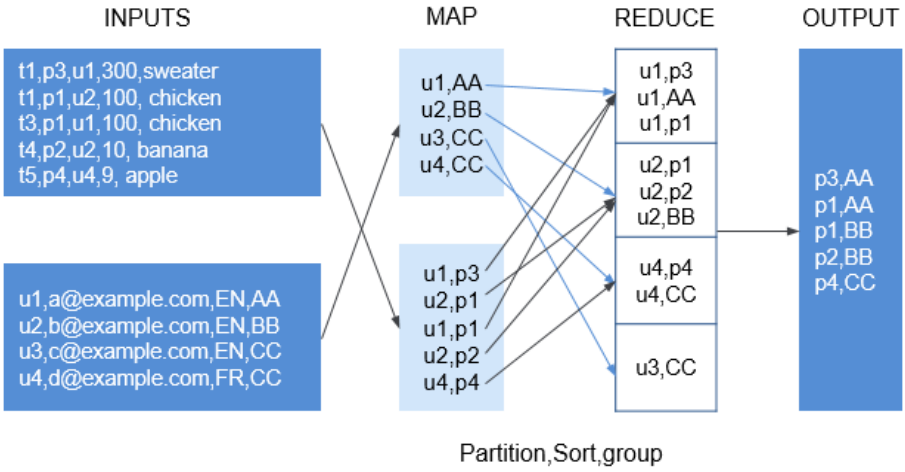
HDFS 开源增强特性：文件块同分布（Colocation）

离线数据汇总统计场景中，Join是一个经常用到的计算功能，在MapReduce中的实现方式大体如下：

1. Map任务分别将两个表文件的记录处理成（Join Key，Value），然后按照Join Key做Hash分区后，送到不同的Reduce任务里去处理。
2. Reduce任务一般使用Nested Loop方式递归左表的数据，并遍历右表的每一行，对于相等的Join Key，处理Join结果并输出。

   以上方式的最大问题在于，由于数据分散在各节点上，所以在Map到Reduce过程中，需要大量的网络数据传输，使得Join计算的性能大大降低，该过程如图6-45所示：

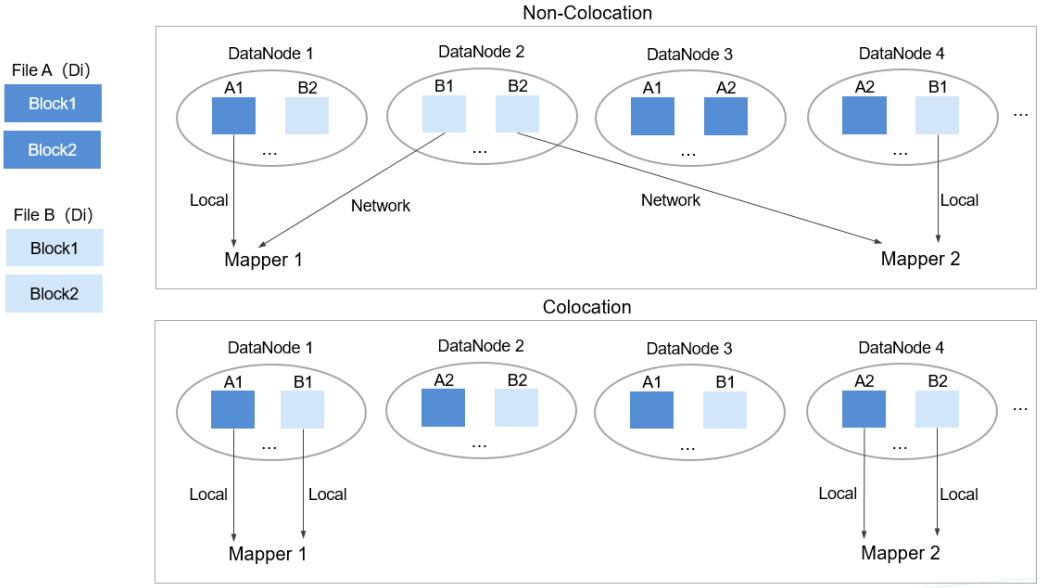
图 6-45 无同分布数据传输流程



由于数据表文件是以HDFS Block方式存放在物理文件系统中，如果能把两个需要Join的文件数据块按Join Key分区后，一一对应地放在同一台机器上，则在Join计算的Reduce过程中无需传递数据，直接在节点本地做Map Join后就能得到结果，性能显著提升。

HDFS数据同分布特性，使得需要做关联和汇总计算的两个文件FileA和FileB，通过指定同一个分布ID，使其所有的Block分布在一起，不再需要跨节点读取数据就能完成计算，极大提高MapReduce Join性能。

图 6-46 无同分布与同分布数据块分布对比

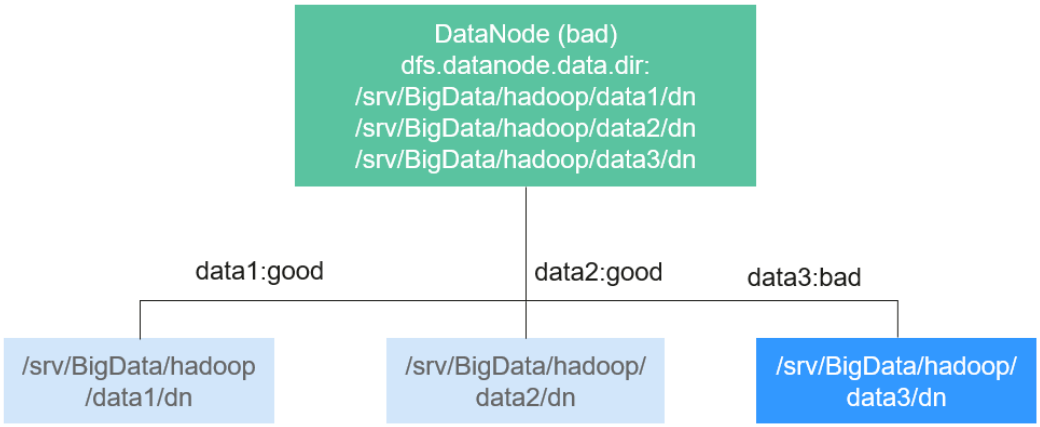


HDFS 开源增强特性：硬盘坏卷设置

在开源版本中，如果为DataNode配置多个数据存放卷，默认情况下其中一个卷损坏，则DataNode将不再提供服务。配置项“dfs.datanode.failed.volumes.tolerated”可以指定失败的个数，小于该个数，DataNode可以继续提供服务。

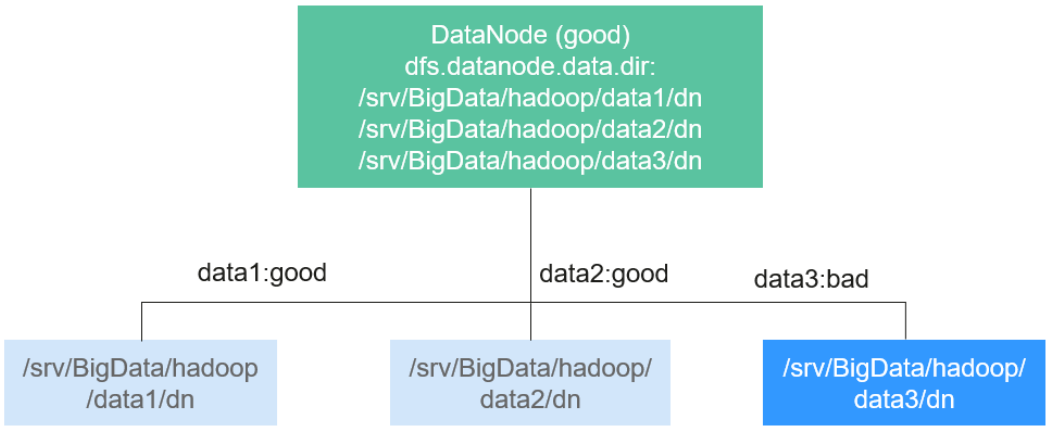
“dfs.datanode.failed.volumes.tolerated”取值范围为-1~DataNode上配置的磁盘卷数，默认值为-1，效果如图6-47所示。

图 6-47 选项设置为 0



例如：某个DataNode中挂载了3个数据存放卷，“dfs.datanode.failed.volumes.tolerated”配置为1，则当该DataNode中的其中一个数据存放卷不能使用的时候，该DataNode会继续提供服务。如图6-48所示。

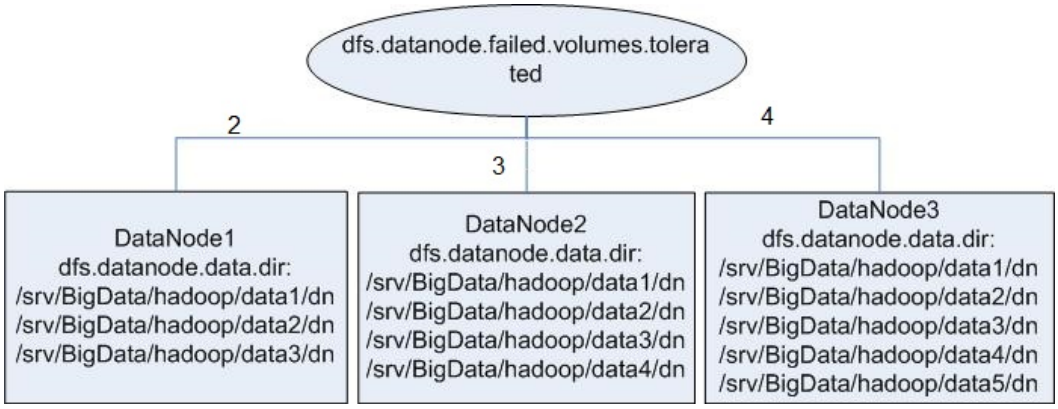
图 6-48 选项设置为 1



这个原生的配置项，存在一定的缺陷。当DataNode的数据存放卷数量不一致的时候，就需要对每个DataNode进行单独配置，而无法配置为所有节点统一生成配置文件，造成用户使用的不便。

例如：集群中存在3个DataNode节点，第一个节点有3个数据目录，第二个节点有4个数据目录，第三个节点有5个数据目录，如果需要实现当节点有一个目录还可用的时候DataNode服务依然可用的效果，就需要如图6-49所示进行设置。

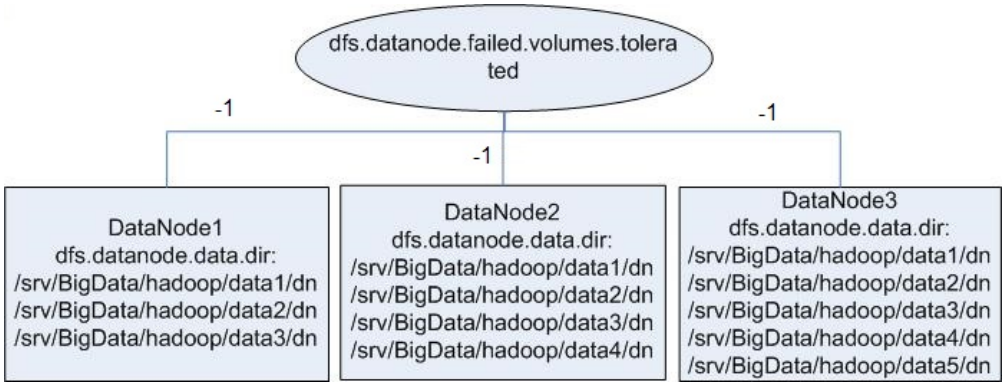
图 6-49 未增强前属性设置



在自研增强版本的HDFS中，对该配置项进行了增强，增加了-1的值选项。当配置成-1的时候，所有DataNode节点只要还有一个数据存放卷，DataNode就能继续提供服务。

所以对于上面提到的例子，该属性的配置将统一成-1，如图6-50所示。

图 6-50 增强后属性配置



HDFS 开源增强特性：HDFS 启动加速

在HDFS中，NameNode启动需要加载元数据文件fsimage，然后等待DataNode完成启动并上报数据块信息。当DataNode上报的数据块信息达到设定百分比时，NameNode退出Safemode，完成启动过程。当HDFS上保存的文件数量达到千万甚至亿级以后，以上两个过程都要耗费大量的时间，致使NameNode的启动过程变得非常漫长。该版本对加载元数据fsimage这一过程进行了优化。

在开源HDFS中，fsimage里保存了所有类型的元数据信息，每一类元数据信息（如文件元数据信息和文件夹元数据信息）分别保存在一个section块里，这些section块在启动时是串行加载的。当HDFS上存储了大量的文件和文件夹时，这两个section的加载就会非常耗时，影响HDFS文件系统的启动时间。HDFS NameNode在生成fsimage时可以将同一类型的元数据信息分段保存在多个section里，当NameNode启动时并行加载fsimage中的section以加快加载速度。

HDFS 开源增强特性：基于标签的数据块摆放策略（HDFS Nodelabel）

用户需要通过数据特征灵活配置HDFS文件数据块的存储节点。通过设置HDFS目录/文件对应一个标签表达式，同时设置每个DataNode对应一个或多个标签，从而给文件的

数据块存储指定了特定范围的DataNode。当使用基于标签的数据块摆放策略，为指定的文件选择DataNode节点进行存放时，会根据文件的标签表达式选择出将要存放的Datanode节点范围，然后在这些Datanode节点范围内，选择出合适的存放节点。

- 支持用户将数据块的各个副本存放在指定具有不同标签的节点，如某个文件的数据块的2个副本放置在标签L1对应节点中，该数据块的其他副本放置在标签L2对应的节点中。
- 支持选择节点失败情况下的策略，如随机从全部节点中选一个。

如图6-51所示。

- /HBase下的数据存储在A，B，D
- /Spark下的数据存储在A，B，D，E，F
- /user下的数据存储在C，D，F
- /user/shl下的数据存储在A，E，F

图 6-51 基于标签的数据块摆放策略样例

HDFS 开源增强特性：HDFS Load Balance

HDFS的现有读写策略主要以数据本地性优先为主，并未考虑节点或磁盘的实际负载情况。HDFS Load Balance功能是基于不同节点的I/O负载情况，在HDFS客户端进行读写操作时，尽可能地选择I/O负载较低的节点进行读写，以此达到I/O负载均衡，以及充分利用集群整体吞吐能力。

写文件时，如果开启写文件的HDFS Load Balance功能，NameNode仍然是根据正常顺序（本地节点—本机架—远端机架）进行DataNode节点的选取，只是在每次选择节点后，如果该节点I/O负载较高，会舍弃并从其他节点中重新选取。

读文件时，Client会向NameNode请求所读Block所在的DataNode列表。NameNode会返回根据网络拓扑距离进行排序的DataNode列表。开启读取的HDFS Load Balance功能时，NameNode会在原先网络拓扑距离排序的基础上，根据每个节点的平均I/O负载情况进行顺序调整，把高I/O负载的节点顺序调整至后面。

HDFS 开源增强特性：HDFS 冷热数据迁移

Hadoop历来主要被用于批量处理大规模的数据。相比处理低时延，批处理应用更关注原始数据处理的吞吐量，因此，目前已有的HDFS模型都运作良好。

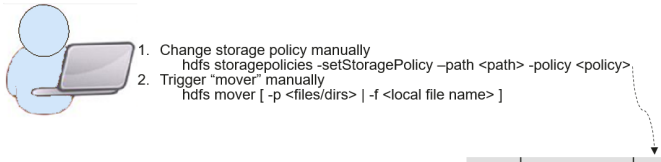
然而，随着技术的发展，Hadoop逐渐被用于以随机I/O访问模式的操作为主的上层应用上，如Hive、HBase等，而这种时延要求较高的场景中，低时延的高速磁盘（如SSD磁盘）可以得到广泛的应用。为了支持这种特性，HDFS现在支持了异构存储类型，这样用户就可以根据自己不同的业务需求场景来选择不同的数据存储类型。

因此，HDFS可以根据数据的热度，选择不同的存储策略。如将HDFS上频繁访问多次的数据被标识为ALL_SSD或HOT，被访问几次的可以标识为WARM，而只有访问1～2次甚至更少的可以被标识为COLD等，如下图为不同的数据热度，可以选择不同的数据存储策略。

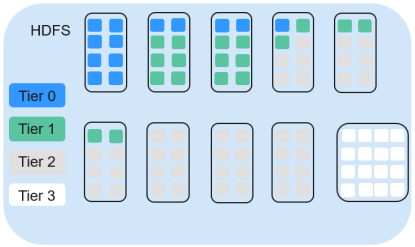
但是，这些高速低时延磁盘，例如SSD磁盘，通常比机械磁盘贵很多。大部分用户希望只有那些经常被访问的热数据才能一直被存储在昂贵的高速磁盘上，而随着数据的访问热度下降以及时间的老化，这些数据应该被迁移到价格低廉的存储介质上。

以详单查询场景作为典型的用例场景，进行说明：当最新详单数据刚刚被导入HDFS上时，会被上层业务人员频繁查询，所以为了提高查询性能，可以将这些详单数据最先导入到SSD磁盘中；但是随着时间的迁移，这些数据逐渐被老化，访问频度越来越低，这时便不适合继续存储在高速硬盘上，需要迁移到廉价的存储介质，节省成本。

目前，如下图所示，HDFS无法很好的支持这些操作，需要自己根据业务类型手动识别数据的热度，并且手动设定数据的存储策略，最后手动触发HDFS Auto Data Movement工具进行数据迁移。



Policy ID	PolicyName	Block Placement (n replacas)	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>



因此，能够基于数据的age自动识别出老化的数据，并将它们迁移到价格低廉的存储介质（如Disk/Archive）上，会给用户节省很高的存储成本，提高数据管理效率。

HDFS Auto Data Movement工具是HDFS冷热数据迁移的核心，根据数据的使用频率自动识别数据冷热设定不同的存储策略。该工具主要支持以下功能：

- 根据数据的age，access time和手动迁移规则，将数据存储策略标识为All_SSD/One_SSD/Hot/Warm/Cold。
- 根据数据age，access time和手动迁移规则，定义区分冷热数据的规则。
- 定义基于age的规则匹配时要采取的行为操作。
  - MARK：表示只会基于age规则标识出数据的冷热度，并设置出对应的存储策略。
  - MOVE：表示基于age规则识别出相应的数据冷热度，并标记出对应的存储策略后，并触发HDFS Auto Data Movement工具进行数据搬迁，调用HDFS冷热数据迁移工具并跨层迁移数据的行为操作。
  - SET_REPL：为文件设置新的副本数的行为操作。
  - MOVE_TO_FOLDER：将文件移动到目标文件夹的行为操作。
  - DELETE：删除文件/目录的行为操作。
  - SET_NODE_LABEL：设置文件节点标签（NodeLabel）的操作。

使用HDFS冷热数据迁移功能，只需要定义age，基于access time的规则。由HDFS冷热数据迁移工具来匹配基于age的规则的数据，设置存储策略和迁移数据。以这种方式，提高了数据管理效率和集群资源效率。

6.11 HetuEngine

6.11.1 HetuEngine 基本原理

HetuEngine 简介

HetuEngine是自研企业级高性能交互式SQL分析及数据虚拟化引擎。与大数据生态无缝融合，实现海量数据秒级交互式查询；支持跨源跨域统一访问，使能数据湖内、湖间、湖仓一站式SQL融合分析。

HetuEngine在兼容开源Trino SQL语法的基础之上，在安全、高可用、易用性等维度提供企业级增强特性，内核能力跟随开源社区持续演进。

更多关于HetuEngine组件操作指导，请参考**使用HetuEngine**。

HetuEngine 结构

HetuEngine包含不同模块，整体结构如**图6-52**所示。

图 6-52 HetuEngine 结构图

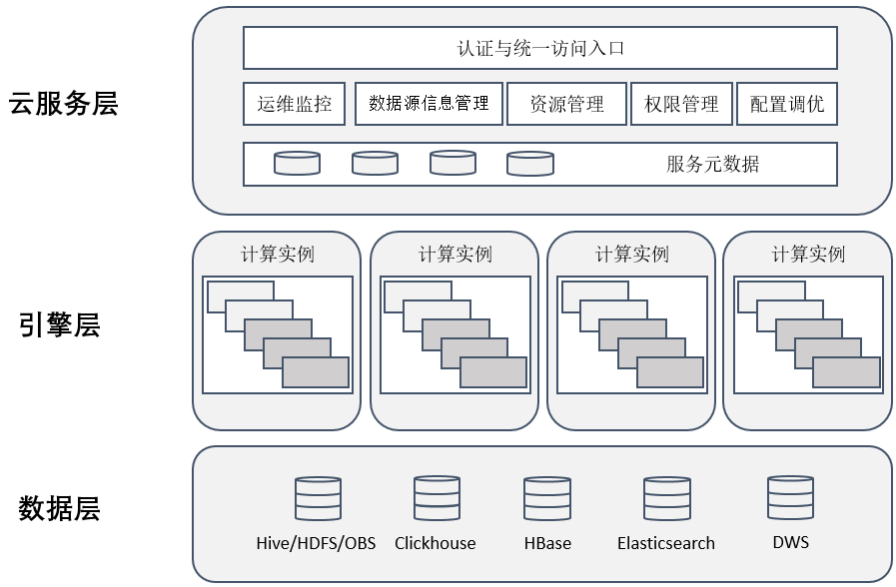


表 6-10 各模块说明

模块名称	常见概念名称	描述
云服务层	HetuEngine CLI/JDBC	HetuEngine的客户端，使用者通过客户端向服务端提交查询请求，然后将执行结果取回并展示。
	HSBroker	HetuEngine的服务管理，用作计算实例的资源管理校验，健康监控与自动维护等。
	HSConsole	对外提供数据源信息管理，计算实例管理，自动化任务的查看等功能的可视化操作界面和RESTful接口。
	HSFabric	提供跨域（DC）高性能安全数据传输。
引擎层	Coordinator	HetuEngine计算实例的管理节点，提供SQL接收、SQL解析、生成执行计划、执行计划优化、分派任务和资源调度等能力。
	Worker	HetuEngine计算实例的工作节点，提供数据源数据并行拉取，分布式SQL计算等能力。

HetuEngine 应用场景

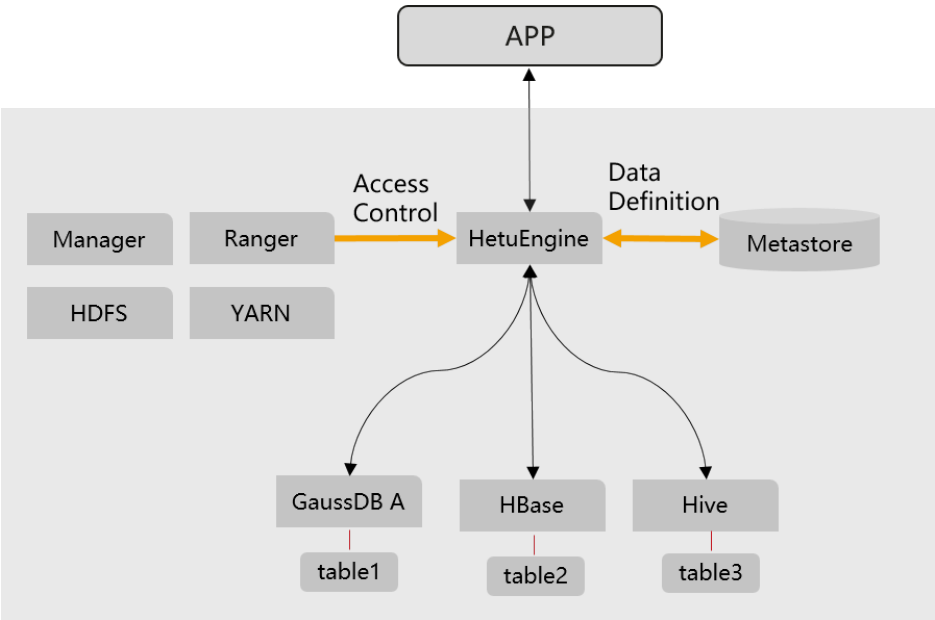
HetuEngine能够支持跨源（多种数据源，如Hive，HBase，GaussDB(DWS)，ClickHouse等），跨域（多个地域或数据中心）的快速联合查询，尤其适用于Hadoop集群（MRS）的Hive、Hudi数据的交互式快速查询场景。

HetuEngine 跨源功能简介

出于管理和信息收集的需要，企业内部会存储海量数据，包括数目众多的各种数据库、数据仓库等，此时会面临数据源种类繁多、数据集结构化混合、相关数据存放分散等困境，导致跨源查询开发成本高，跨源复杂查询耗时长。

HetuEngine提供了统一标准SQL实现跨源协同分析，简化跨源分析操作。

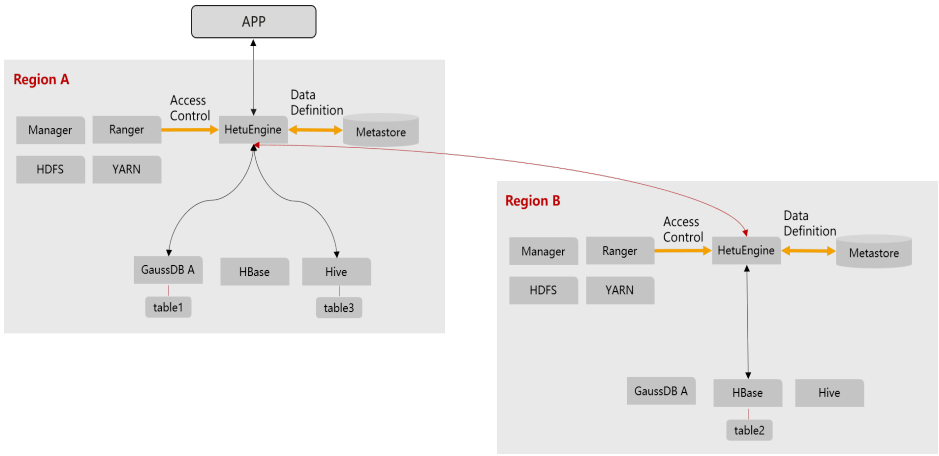
图 6-53 HetuEngine 跨源功能示意



HetuEngine 跨域功能简介

HetuEngine提供统一标准SQL对分布于多个地域（或数据中心）的多种数据源实现高效访问，屏蔽数据在结构、存储及地域上的差异，实现数据与应用的解耦。

图 6-54 HetuEngine 跨域功能示意



6.11.2 HetuEngine 与其他组件的关系

HetuEngine安装依赖MRS集群，其中直接依赖的组件如表6-11所示。

表 6-11 HetuEngine 依赖的组件

名称	描述
HDFS	Hadoop分布式文件系统（Hadoop Distributed File System），提供高吞吐量的数据访问，适合大规模数据集方面的应用。
Hive	建立在Hadoop基础上的开源的数据仓库，提供类似SQL的Hive Query Language语言操作结构化数据存储服务和基本的数据分析服务。
ZooKeeper	提供分布式、高可用性的协调服务能力。帮助系统避免单点故障，从而建立可靠的应用程序。
KrbServer	密钥的管理中心，负责票据的分发。
Yarn	资源管理系统，它是一个通用的资源模块，可以为各类应用程序进行资源管理和调度。
DBService	高可用性的关系型数据库存储系统，提供元数据的备份与恢复功能。

6.12 Hive

6.12.1 Hive 基本原理

Hive是建立在Hadoop上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在Hadoop中的大规模数据的机制。Hive定义了简单的类SQL查询语言，称为HQL，它允许熟悉SQL的用户查询数据。Hive的数据计算依赖于MapReduce、Spark、Tez。

使用新的执行引擎Tez代替原先的MapReduce，性能有了显著提升。Tez可以将多个有依赖的作业转换为一个作业（这样只需写一次HDFS，且中间节点较少），从而大大提升DAG作业的性能。

Hive主要特点如下：

- 海量结构化数据分析汇总。
- 将复杂的MapReduce编写任务简化为SQL语句。
- 灵活的数据存储格式，支持JSON，CSV，TEXTFILE，RCFILE，SEQUENCEFILE，ORC（Optimized Row Columnar）这几种存储格式。

Hive体系结构：

- 用户接口：用户接口主要有CLI，Client和WebUI。其中最常用的是CLI，CLI启动的时候，会同时启动一个Hive副本。Client是Hive的客户端，用户连接至Hive Server。在启动Client模式的时候，需要指出Hive Server所在节点，并且在该节点启动Hive Server。WebUI是通过浏览器访问Hive。MRS仅支持Client方式访问Hive，使用操作请参考**快速使用Hive进行数据分析**，应用开发请参考**Hive应用开发简介**。
- 元数据存储：Hive将元数据存储在数据库中，如MySQL、Derby。Hive中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

更多关于Hive组件操作指导，请参考**使用Hive**。

Hive 结构

Hive为单实例的服务进程，提供服务的原理是将HQL编译解析成相应的MapReduce或者HDFS任务，**图6-55**为Hive的结构概图。

图 6-55 Hive 结构

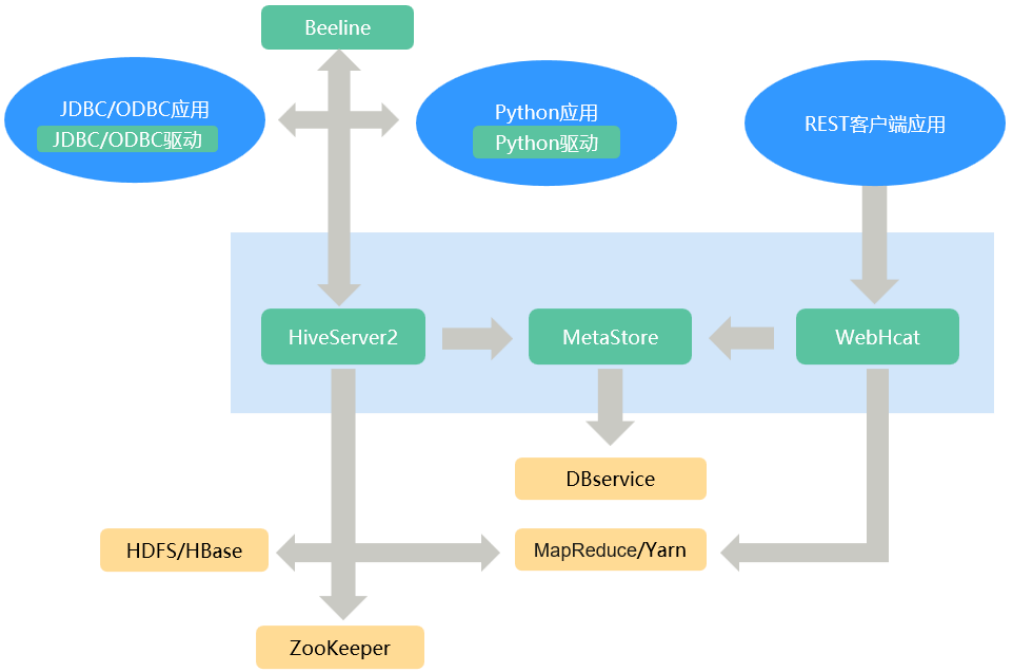
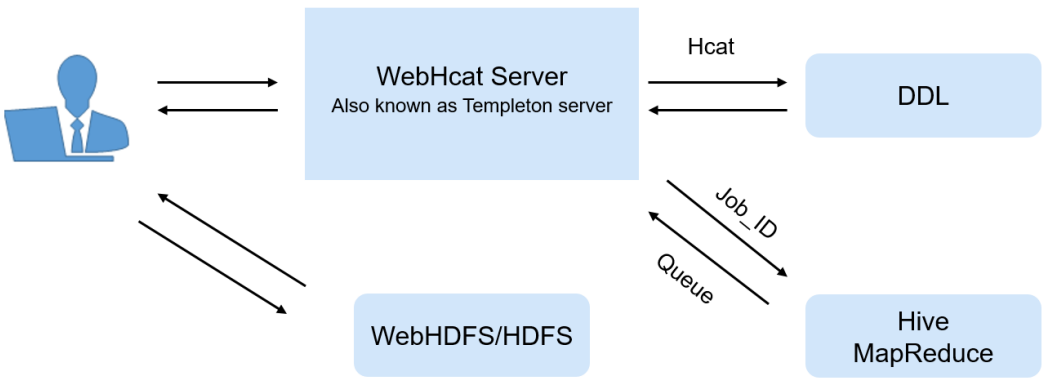


表 6-12 模块说明

名称	说明
HiveServer	一个集群内可部署多个HiveServer，负荷分担。对外提供Hive数据库服务，将用户提交的HQL语句进行编译，解析成对应的Yarn任务或者HDFS操作，从而完成数据的提取、转换、分析。
MetaStore	• 一个集群内可部署多个MetaStore，负荷分担。提供Hive的元数据服务，负责Hive表的结构和属性信息读、写、维护和修改。 • 提供Thrift接口，供HiveServer、Spark、WebHCat等MetaStore客户端来访问，操作元数据。
WebHCat	一个集群内可部署多个WebHCat，负荷分担。提供Rest接口，通过Rest执行Hive命令，提交MapReduce任务。
Hive客户端	包括人机交互命令行Beeline、提供给JDBC应用的JDBC驱动、提供给Python应用的Python驱动、提供给MapReduce的HCatalog相关JAR包。
ZooKeeper集群	ZooKeeper作为临时节点记录各HiveServer实例的IP地址列表，客户端驱动连接ZooKeeper获取该列表，并根据路由机制选取对应的HiveServer实例。
HDFS/HBase集群	Hive表数据存储在HDFS集群中。
MapReduce/Yarn集群	提供分布式计算服务：Hive的大部分数据操作依赖MapReduce/Yarn集群，HiveServer的主要功能是将HQL语句转换成分布式计算任务，从而完成对海量数据的处理。

HCatalog建立在Hive Metastore之上，具有Hive的DDL能力。从另外一种意义上说，HCatalog还是Hadoop的表和存储管理层，它使用户能够通过使用不同的数据处理工具（比如MapReduce），更轻松地在网格上读写HDFS上的数据，HCatalog还能为这些数据处理工具提供读写接口，并使用Hive的命令行接口发布数据定义和元数据探索命令。此外，经过封装这些命令，WebHCat Server还对外提供了RESTful接口，如图6-56所示。

图 6-56 WebHCat 的逻辑架构图



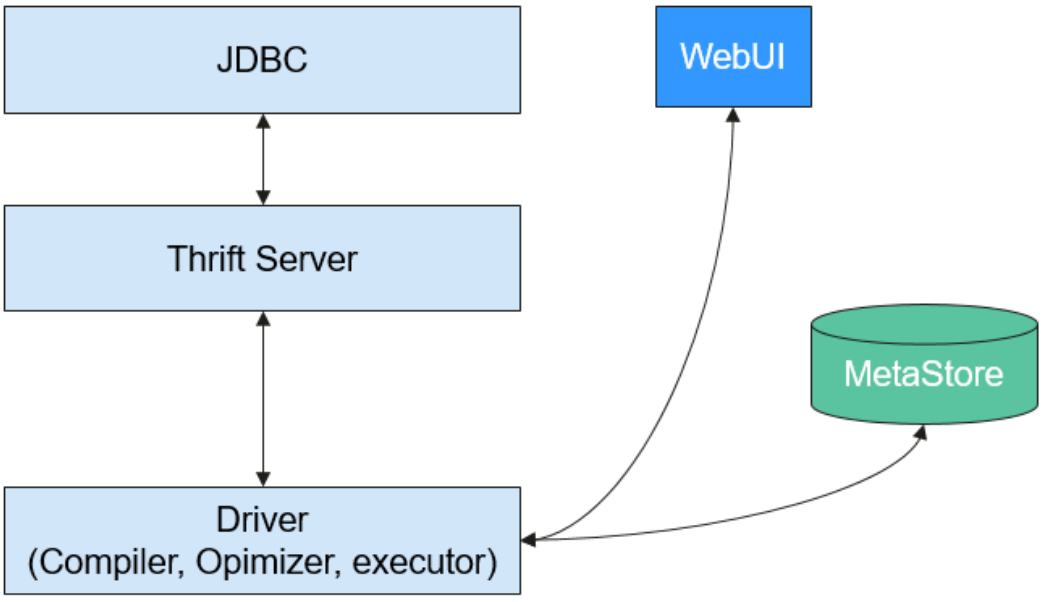
Hive 原理

Hive作为一个基于HDFS和MapReduce架构的数据仓库，其主要能力是通过对HQL（Hive Query Language）编译和解析，生成并执行相应的MapReduce任务或者HDFS操作。Hive与HQL相关信息，请参考**HQL 语言手册**。

图6-57为Hive的结构简图。

- **Metastore**：对表，列和Partition等的元数据进行读写及更新操作，其下层为关系型数据库。
- **Driver**：管理HQL执行的生命周期并贯穿Hive任务整个执行期间。
- **Compiler**：编译HQL并将其转化为一系列相互依赖的Map/Reduce任务。
- **Optimizer**：优化器，分为逻辑优化器和物理优化器，分别对HQL生成的执行计划和MapReduce任务进行优化。
- **Executor**：按照任务的依赖关系分别执行Map/Reduce任务。
- **ThriftServer**：提供thrift接口，作为JDBC的服务端，并将Hive和其他应用程序集成起来。
- **Clients**：包含WebUI和JDBC接口，为用户访问提供接口。

图 6-57 Hive 结构



6.12.2 Hive CBO 原理介绍

Hive CBO 原理介绍

CBO，全称是Cost Based Optimization，即基于代价的优化器。

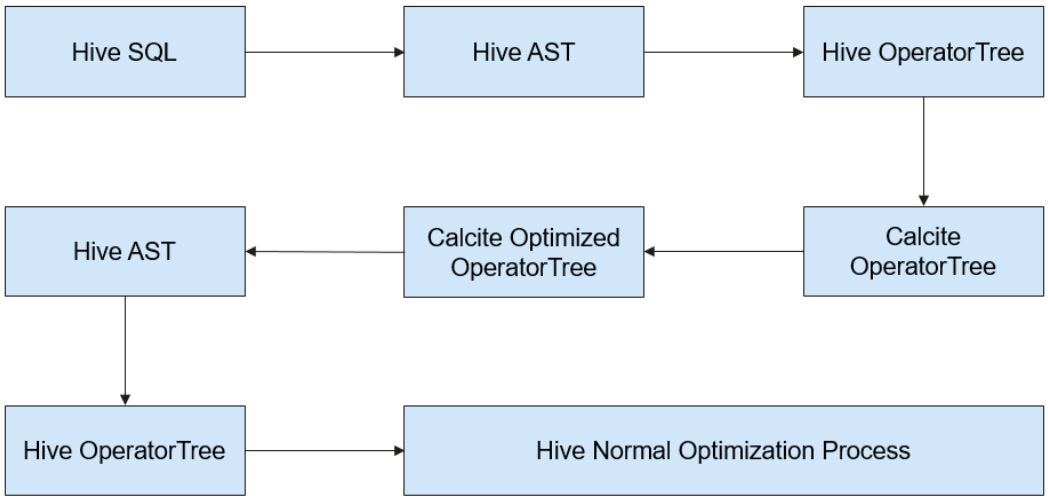
其优化目标是：

在编译阶段，根据查询语句中涉及到的表和查询条件，计算出产生中间结果少的高效join顺序，从而减少查询时间和资源消耗。

Hive中实现CBO的总体过程如下：

Hive使用开源组件Apache Calcite实现CBO。首先SQL语句转化成Hive的AST，然后转成Calcite可以识别的RelNodes。Calcite将RelNode中的Join顺序调整后，再由Hive将RelNode转成AST，继续Hive的逻辑优化和物理优化过程。流程图如图6-58所示：

图 6-58 实现流程图



Calcite调整Join顺序的具体过程如下：

1. 针对所有参与Join的表，依次选取一个表作为第一张表。
2. 依据选取的第一张表，根据代价选择第二张表，第三张表。由此可以得到多个不同的执行计划。
3. 计算出代价最小的一个计划，作为最终的顺序优化结果。

代价的具体计算方法：

当前版本，代价的衡量基于Join出来的数据条数：Join出来的条数越少，代价越小。Join条数的多少，取决于参与Join的表的选择率。表的数据条数，取自表级别的统计信息。

过滤条件过滤后的条数，由列级别的统计信息，max，min，以及NDV（Number of Distinct Values）来估算出来。

例如存在一张表table_a，其统计信息如下：数据总条数1000000，NDV 50，查询条件如下：

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

则估算查询的最终条数为1000000 * 1/50 = 20000条，选择率为2%。

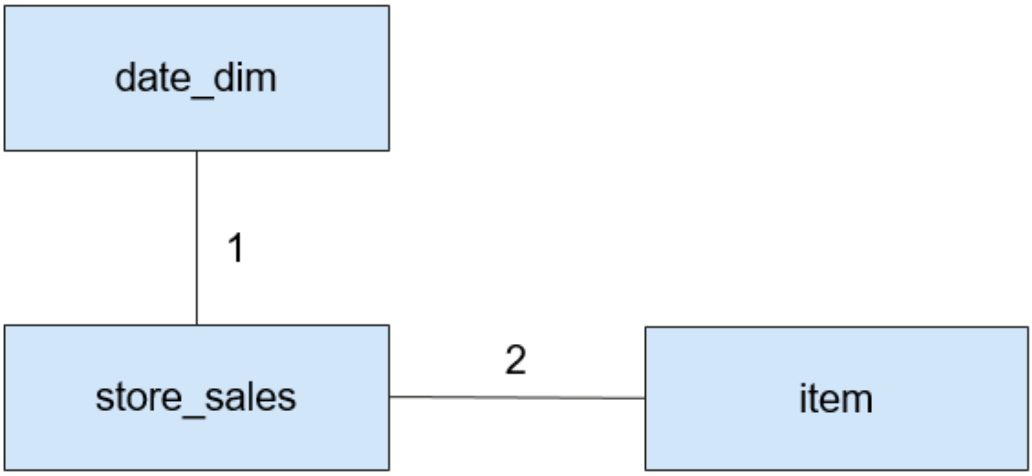
以下以TPC-DS Q3为例来介绍CBO是如何调整Join顺序的。

```
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;
```

语句解释：这个语句由三张表来做Inner join，其中store_sales是事实表，有约2900000000条数据，date_dim是维度表，有约73000条数据，item是维度表，有约18000条数据。每一个表上都有过滤条件，其Join关系如所图6-59示：

图 6-59 Join 关系



CBO应该先选择能起到更好过滤效果的表来Join。

通过分析min，max，NDV，以及数据条数。CBO估算出不同维度表的选择率，详情如表6-13所示。

表 6-13 数据过滤

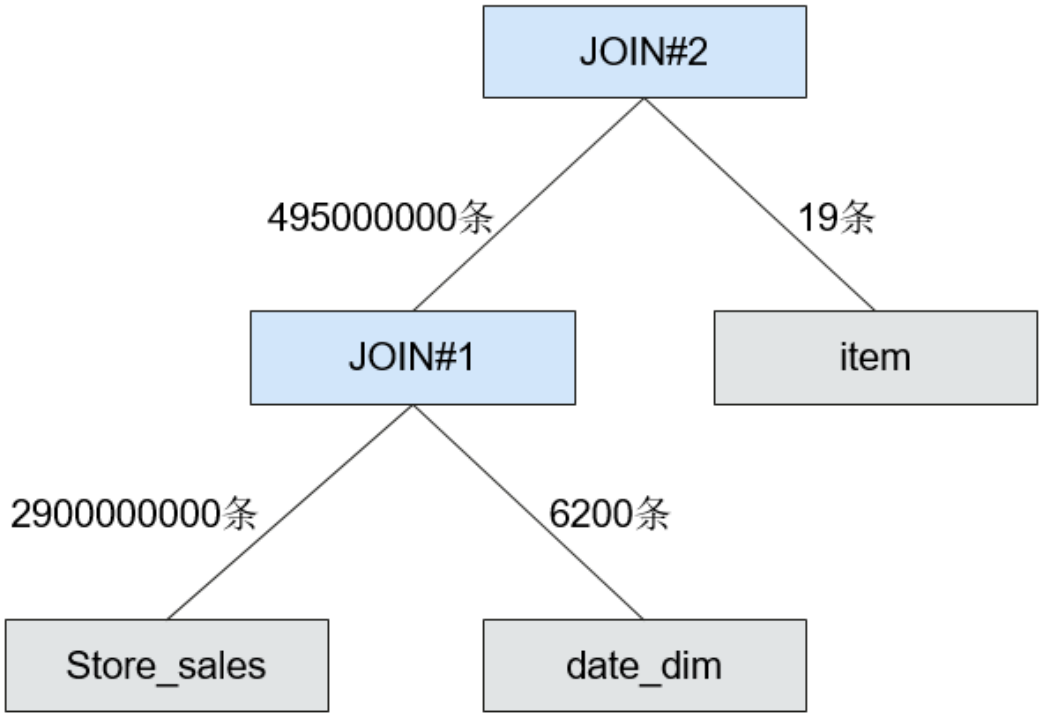
表名	原始数据条数	过滤后数据条数	选择率
date_dim	73000	6200	8.5%
item	18000	19	0.1%

上述表格获取到原始表的数据条数，估算出过滤后的数据条数后，计算出选择率=过滤后条数/原始条数。

从上表可以看出，item表具有较好的过滤效果，因此CBO将item表的Join顺序提前。

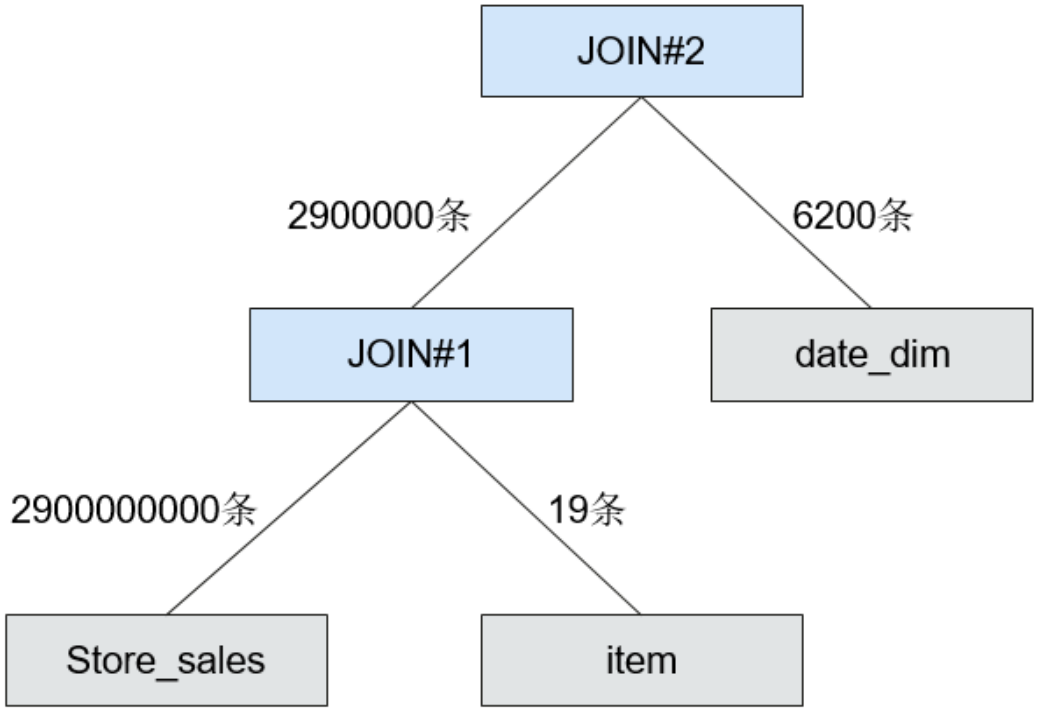
CBO未开启时的Join示意图如图6-60所示：

图 6-60 未开启 CBO



CBO开启后的Join示意图如图6-61所示：

图 6-61 开启 CBO



可以看出，优化后中间结果由495000000条减少到了2900000条，执行时间也大幅减少。

6.12.3 Hive 与其他组件的关系

Hive 与 HDFS 组件的关系

Hive是Apache的Hadoop项目的子项目，Hive利用HDFS作为其文件存储系统。Hive通过解析和计算处理结构化的数据，Hadoop HDFS则为Hive提供了高可靠性的底层存储支持。Hive数据库中的所有数据文件都可以存储在Hadoop HDFS文件系统上，Hive所有的数据操作也都是通过Hadoop HDFS接口进行的。

Hive 与 MapReduce 组件的关系

Hive的数据计算依赖于MapReduce。MapReduce也是Apache的Hadoop项目的子项目，它是一个基于Hadoop HDFS分布式并行计算框架。Hive进行数据分析时，会将用户提交的HQL语句解析成相应的MapReduce任务并提交MapReduce执行。

Hive 与 Tez 的关系

Tez是Apache的开源项目，它是一个支持有向无环图的分布式计算框架，Hive使用Tez引擎进行数据分析时，会将用户提交的HQL语句解析成相应的Tez任务并提交Tez执行。

Hive 与 DBService 的关系

Hive的MetaStore（元数据服务）处理Hive的数据库、表、分区等的结构和属性信息（即Hive的元数据），这些信息需要存放在一个关系型数据库中，由MetaStore管理和处理。在产品中，Hive的元数据由DBService组件存储和维护，由Metadata组件提供元数据服务。

Hive 与 Spark 的关系

Hive支持使用Spark作为执行引擎，当执行引擎切换为Spark后，客户端下发的Hive SQL在Hive端进行逻辑层处理和生成物理执行计划，并将执行计划转换成RDD语义下的DAG，最后将DAG作为Spark的任务提交到Spark集群上进行计算，并合理利用Spark分布式内存计算能力，提高了Hive查询效率。

6.12.4 Hive 开源增强特性

Hive 开源增强特性：支持 HDFS Colocation

HDFS Colocation（同分布）是HDFS提供的数据分布控制功能，利用HDFS Colocation接口，可以将存在关联关系或者可能进行关联操作的数据存放在相同的存储节点上。

Hive支持HDFS的Colocation功能，即在创建Hive表时，通过设置表文件分布的locator信息，可以将相关表的数据文件存放在相同的存储节点上，从而使后续的多表关联的数据计算更加方便和高效。

Hive 开源增强特性：支持列加密功能

Hive支持对表的某一列或者多列进行加密。在创建Hive表时，可以指定要加密的列和加密算法。当使用insert语句向表中插入数据时，即可将对应的列进行加密。Hive列加密不支持视图以及Hive over HBase场景。

Hive列加密机制目前支持的加密算法有两种，具体使用的算法在建表时指定。

- AES（对应加密类名称为：org.apache.hadoop.hive.serde2.AESRewriter）
- SM4（也称为SMS4，对应加密类名称为：org.apache.hadoop.hive.serde2.SMS4Rewriter）

Hive 开源增强特性：支持 HBase 删除功能

由于底层存储系统的原因，Hive并不能支持对单条表数据进行删除操作，但在Hive on HBase功能中，MRS解决方案中的Hive提供了对HBase表的单条数据的删除功能，通过特定的语法，Hive可以将自己在HBase表中符合条件的一条或者多条数据清除。

Hive 开源增强特性：支持行分隔符

通常情况下，Hive以文本文件存储的表会以回车作为其行分隔符，即在查询过程中，以回车符作为一行表数据的结束符。

但某些数据文件并不是以回车分隔的规则文本格式，而是以某些特殊符号分隔其规则文本。

MRS Hive支持指定不同的字符或字符组合作为Hive文本数据的行分隔符。

Hive 开源增强特性：支持基于 HTTPS/HTTP 协议的 REST 接口切换

WebHCat为Hive提供了对外可用的REST接口，开源社区版本默认使用HTTP协议。

MRS Hive支持使用更安全的HTTPS协议，并且可以在两种协议间自由切换。

Hive 开源增强特性：支持开启 Transform 功能

Hive开源社区版本禁止Transform功能。 MRS Hive提供配置开关，Transform功能默认为禁止，与开源社区版本保持一致。

用户可修改配置开关，开启Transform功能，当开启Transform功能时，存在一定的安全风险。

Hive 开源增强特性：支持创建临时函数不需要 ADMIN 权限的功能

Hive开源社区版本创建临时函数需要用户具备ADMIN权限。MRS Hive提供配置开关，默认为创建临时函数需要ADMIN权限，与开源社区版本保持一致。

用户可修改配置开关，实现创建临时函数不需要ADMIN权限。

Hive 开源增强特性：支持数据库授权

Hive开源社区版本只支持数据库的拥有者在数据库中创建表。MRS Hive支持授予用户在数据库中创建表“CREATE”和查询表“SELECT”权限。当授予用户在数据库中查询的权限之后，系统会自动关联数据库中所有表的查询权限。

Hive 开源增强特性：支持列授权

Hive开源社区版本只支持表级别的权限控制。MRS Hive支持列级别的权限控制，可授予用户列级别权限，例如查询“SELECT”、插入“INSERT”、修改“UPDATE”权限。

6.13 Hudi

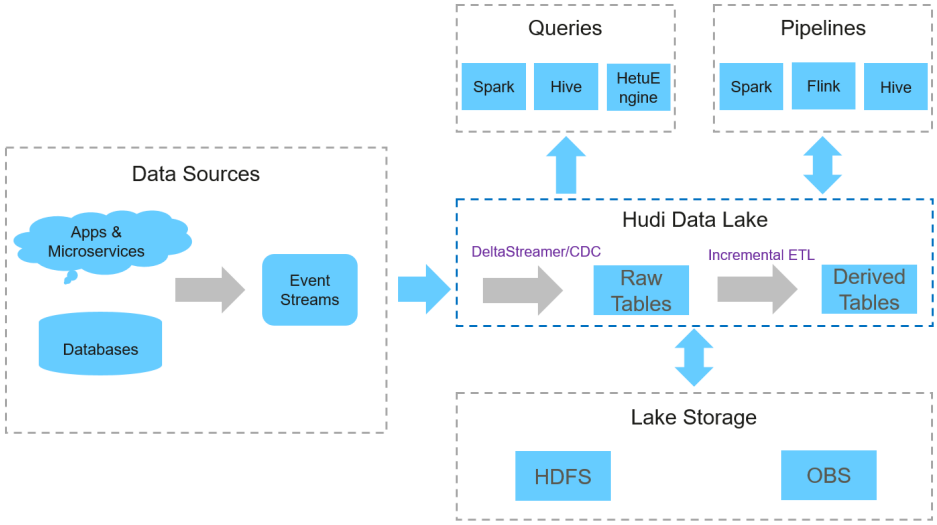
Hudi是一种数据湖的存储格式，在Hadoop文件系统之上提供了更新数据和删除数据的能力以及消费变化数据的能力。支持多种计算引擎，提供IUD接口，在HDFS的数据集上提供了插入更新和增量拉取的功能。

更多关于Hudi组件操作指导，请参考**使用Hudi**。

说明

如需使用Hudi，请确保MRS集群内已安装Spark/Spark2x服务。

图 6-62 Hudi 基本架构



Hudi 特性

- ACID事务能力，支持实时入湖和批量入湖。
- 多种视图能力（读优化视图/增量视图/实时视图），支持快速数据分析。
- MVCC设计，支持数据版本回溯。
- 自动管理文件大小和布局，以优化查询性能准实时摄取，为查询提供最新数据。
- 支持并发读写，基于snapshot的隔离机制实现写入时可读取。
- 支持原地转表，将存量的历史表转换为Hudi数据集。

Hudi 关键技术和优势

- 可插拔索引机制：Hudi提供多种索引机制，可以快速完成对海量数据的更新和删除操作。
- 良好的生态支持：Hudi支持多种数据引擎接入包括Hive、Spark、Flink。

Hudi 支持两种表类型

- Copy On Write

  写时复制表也简称cow表，使用parquet文件存储数据，内部的更新操作需要通过重写原始parquet文件完成。

  - 优点：读取时，只读取对应分区的一个数据文件即可，较为高效。
  - 缺点：数据写入的时候，需要复制一个先前的副本再在其基础上生成新的数据文件，这个过程比较耗时。且由于耗时，读请求读取到的数据相对就会滞后。
- Merge On Read

  读时合并表也简称mor表，使用列格式parquet和行格式Avro两种方式混合存储数据。其中parquet格式文件用于存储基础数据，Avro格式文件（也可叫做log文件）用于存储增量数据。
  - 优点：由于写入数据先写delta log，且delta log较小，所以写入成本较低。
  - 缺点：需要定期合并整理compact，否则碎片文件较多。读取性能较差，因为需要将delta log和老数据文件合并。

Hudi 支持三种视图，针对不同场景提供相应的读能力

- Snapshot View

  实时视图：该视图提供当前hudi表最新的快照数据，即一旦有最新的数据写入hudi表，通过该视图就可以查出刚写入的新数据。

  cow表和mor均支持这种视图能力。
- Incremental View

  增量视图：该视图提供增量查询的能力，可以查询指定COMMIT之后的增量数据，可用于快速拉取增量数据。

  cow表支持该种视图能力， mor表也可以支持该视图，但是一旦mor表完成compact操作其增量视图能力消失。
- Read Optimized View

  读优化视图：该视图只会提供最新版本的parquet文件中存储的数据。

  该视图在cow表和mor表上表现不同：

  对于cow表，该视图能力和实时视图能力是一样的（cow表只用parquet文件存数据）。

  对于mor表，仅访问基本文件，提供给定文件片自上次执行compact操作以来的数据， 可简单理解为该视图只会提供mor表parquet文件存储的数据，log文件里面的数据将被忽略。 该视图数据并不一定是最新的，但是mor表一旦完成compact操作，增量log数据被合入到了base数据里面，这个时候该视图和实时视图能力一样。

6.14 Hue

6.14.1 Hue 基本原理

Hue是一组WEB应用，用于和MRS大数据组件进行交互，能够帮助用户浏览HDFS，进行Hive查询，启动MapReduce任务等，它承载了与所有MRS大数据组件交互的应用。

Hue主要包括了文件浏览器和查询编辑器的功能：

- 文件浏览器能够允许用户直接通过界面浏览以及操作HDFS的不同目录；
- 查询编辑器能够编写简单的SQL，查询存储在Hadoop之上的数据。例如HDFS，HBase，Hive。 用户可以方便地创建、管理、执行SQL，并且能够以Excel的形式下载执行的结果。

通过Hue可以在界面针对组件进行以下操作：

- HDFS：
  - 查看、创建、管理、重命名、移动、删除文件/目录。
  - 上传、下载文件。
  - 搜索文件、目录、文件所有人、所属用户组；修改文件以及目录的属主和权限。
  - 手动配置HDFS目录存储策略，配置动态存储策略等操作。
- Hive：
  - 编辑、执行SQL/HQL语句；保存、复制、编辑SQL/HQL模板；解释SQL/HQL语句；保存SQL/HQL语句并进行查询。
  - 数据库展示，数据表展示。
  - 支持多种Hadoop存储。
  - 通过Metastore对数据库及表和视图进行增删改查等操作。

    说明

    如果使用IE浏览器访问Hue界面来执行HQL，由于浏览器存在的功能问题，将导致执行失败。建议使用兼容的浏览器，例如Google Chrome浏览器。
- Impala：
  - 编辑、执行SQL/HQL语句；保存、复制、编辑SQL/HQL模板；解释SQL/HQL语句；保存SQL/HQL语句并进行查询。
  - 数据库展示，数据表展示。
  - 支持多种Hadoop存储。
  - 通过Metastore对数据库及表和视图进行增删改查等操作。

    说明

    如果使用IE浏览器访问Hue界面来执行HQL，由于浏览器存在的功能问题，将导致执行失败。建议使用兼容的浏览器，例如Google Chrome浏览器。
- MapReduce：查看集群中正在执行和已经完成的MR任务，包括它们的状态、起始结束时间、运行日志等。
- Oozie：提供了Oozie作业管理器功能，使用户可以通过界面图形化的方式使用Oozie。
- ZooKeeper：提供了ZooKeeper浏览器功能，使用户可以通过界面图形化的方式查看ZooKeeper。

有关Hue的详细信息，请参见：**http://gethue.com/**。

更多关于Hue组件操作指导，请参考**使用Hue**。

Hue 结构

Hue是建立在Django Python（开放源代码的Web应用框架）的Web框架上的Web应用程序，采用了MTV（模型M-模板T-视图V）的软件设计模式。

Hue由“Supervisor Process”和“WebServer”构成，“Supervisor Process”是Hue的核心进程，负责应用进程管理。“Supervisor Process”和“WebServer”通过“THRIFT/REST”接口与WebServer上的应用进行交互，如**图6-63**所示。

图 6-63 Hue 架构示意图

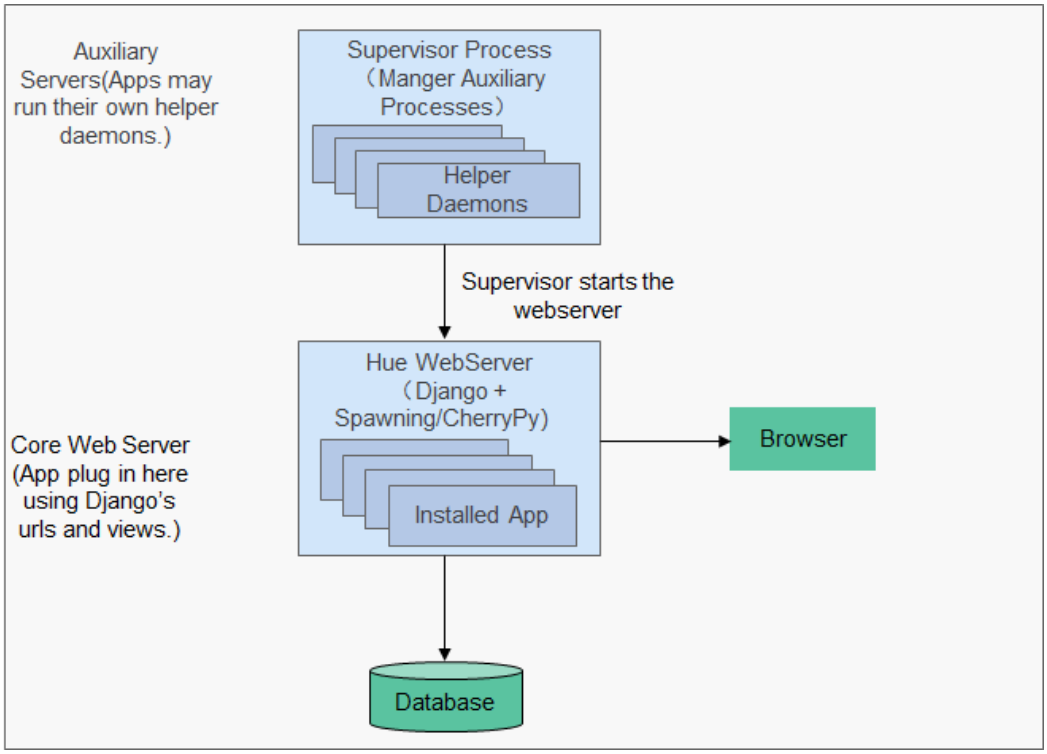


图6-63中各部分的功能说明如表6-14所示。

表 6-14 结构图说明

名称	描述
Supervisor Process	Supervisor负责WebServer上APP的进程管理：启动、停止、监控等。
Hue WebServer	通过Django Python的Web框架提供如下功能。 • 部署APPs。 • 提供图形化用户界面。 • 与数据库连接，存储APP的持久化数据。

6.14.2 Hue 与其他组件的关系

Hue 与 Hadoop 集群的关系

Hue与Hadoop集群的交互关系如图6-64所示。

图 6-64 Hue 与 Hadoop 集群

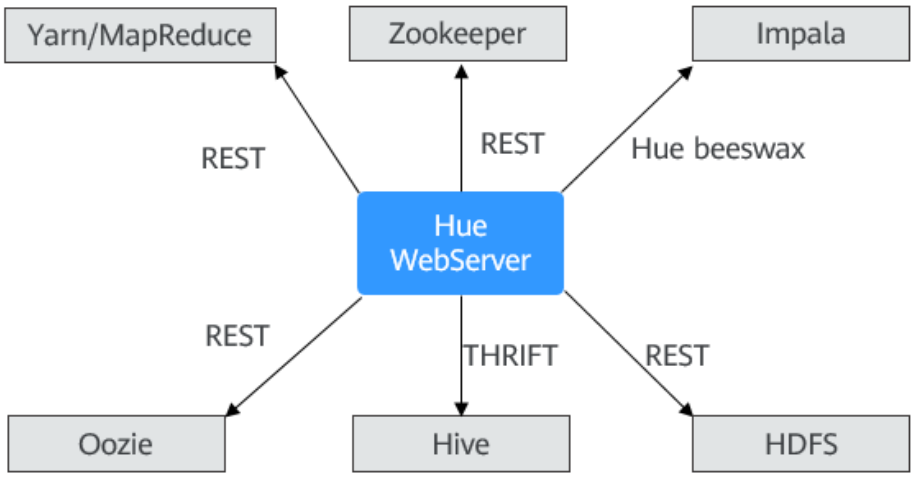


表 6-15 Hue 与其它组件的关系

名称	描述
HDFS	HDFS提供REST接口与Hue交互，用于查询、操作HDFS文件。 在Hue把用户请求从用户界面组装成接口数据，通过调用REST接口调用HDFS，通过浏览器返回结果呈现给用户。
Hive	Hive提供THRIFT接口与Hue交互，用于执行Hive SQL、查询表元数据。 在Hue界面编辑HQL语句，通THRIFT接口提交HQL语句到HiveServer执行，同时把执行通过浏览器呈现给用户。
Yarn/MapReduce	MapReduce提供REST与Hue交互，用于查询Yarn作业信息。 进入Hue页面，输入筛选条件参数，UI将参数发送到后台，Hue通过调用MapReduce（MR1/MR2-YARN）提供的REST接口，获取任务运行的状态，起始结束时间、运行日志等信息。
Oozie	Oozie提供REST接口与Hue交互，用于创建工作流、Coordinator、Bundle，以及它们的任务管理和监控。 在Hue前端提供图形化工作流、Coordinator、Bundle编辑器，Hue调用Oozie REST接口对工作流、Coordinator、Bundle进行创建、修改、删除、提交、监控。
ZooKeeper	ZooKeeper提供REST接口与Hue交互，用于查询ZooKeeper节点信息。 在Hue前端显示ZooKeeper节点信息，Hue调用ZooKeeper REST接口获取这些节点信息。
Impala	Impala提供Hue beeswax接口与Hue交互，用于执行Hive SQL、查询表元数据。 在Hue界面编辑HQL语句，通Hue beeswax接口提交HQL语句到HiveServer执行，同时把执行结果通过浏览器呈现给用户。

6.14.3 Hue 开源增强特性

Hue 开源增强特性

- 存储策略定义。HDFS文件存储在多种等级的存储介质中，有不同的副本数。本特性可以手工设置HDFS目录的存储策略，或者根据HDFS文件最近访问时间和最近修改时间，自动调整文件存储策略、修改文件副本数、移动文件所在目录、自动删除文件，以便充分利用存储的性能和容量。
- MR引擎。用户执行Hive SQL可以选择使用MR引擎执行。
- 可靠性增强。Hue自身主备部署。Hue与HDFS、Oozie、Hive、Yarn等对接时，支持Failover或负载均衡工作模式。

6.15 Impala

Impala

Impala直接对存储在HDFS、HBase或对象存储服务（OBS）中的Hadoop数据提供快速、交互式SQL查询。除了使用相同的统一存储平台之外，Impala还使用于Apache Hive相同的元数据，SQL语法（Hive SQL），ODBC驱动程序和用户界面（Hue中的Impala查询UI）。这为实时或面向批处理的查询提供了一个熟悉且统一的平台。作为查询大数据的工具的补充，Impala不会替代基于MapReduce构建的批处理框架，例如Hive。基于MapReduce构建的Hive和其他框架最适合长时间运行的批处理作业。

Impala主要特点如下：

- 支持Hive查询语言（HQL）中大多数的SQL-92功能，包括SELECT，JOIN和聚合函数。
- HDFS，HBase和对象存储服务（OBS）存储，包括：
  - HDFS文件格式：基于分隔符的Text file，Parquet，Avro，SequenceFile和RCFile。
  - 压缩编解码器：Snappy，GZIP，Deflate，BZIP。
- 常见的数据访问接口包括：
  - JDBC驱动程序。
  - ODBC驱动程序。
  - Hue beeswax和Impala查询UI。
- Impala-shell命令行接口。
- 支持Kerberos身份认证。

Impala主要应用于实时查询数据的离线分析（如日志分析，集群状态分析）、大规模的数据挖掘（用户行为分析，兴趣分区，区域展示）等场景。

有关Impala的详细信息，请参见**https://impala.apache.org/impala-docs.html**。

更多关于Impala组件操作指导，请参考**使用Impala**。

Impala由Impalad、StateStore、Catalog 3个角色组成。

Impala Daemon

Impala daemon的进程名为Impalad，是Impala的核心进程。

Impalad关键功能如下：

- 运行在所有的数据节点上。
- 读写数据文件。
- 接收来自于Impala-shell命令、Hue、JDBC或者ODBC等客户端的查询请求。
- 可以并行执行来自集群中其他节点的查询请求，将中间结果返回给调度节点。
- 可以调用节点将结果返回给客户端。

Impalad进程通过持续的和StateStore通信来确认自己所在的节点是否健康和是否可以接受新的任务请求。

Impala StateStore

负责检查Impala的所有进程健康状态管理进程，进程名为statestored，当有Impalad的进程因硬件失败、网络错误、软件原因或者其他原因下线时，StateStore负责通知到其他的Impalad进程，避免请求分发到不可用的节点上。

Impala Catalog Service

负责Impala的元数据管理，进程名为catalogd，将元数据的变化发送到所有的Impalad进程。当创建表、加载数据或者其他的一些从Hive发起的操作后，Impala查询之前需要在Impalad上执行REFRESH或者INVALIDATE METADATA刷新Catalog上缓存的元数据信息。如果元数据变化是通过Impala执行的，则不需要执行刷新。

Impala 与其他组件的关系

- Impala与HDFS间的关系

  Impala默认利用HDFS作为其文件存储系统。Impala通过解析和计算处理结构化的数据，Hadoop HDFS则为Impala提供了高可靠性的底层存储支持。使用Impala将无需移动HDFS中的数据并且提供更快的访问。

- Impala与Hive间的关系

  Impala使用Hive的元数据、ODBC驱动程序和SQL语法。与Hive不同，Impala不基于MapReduce算法，它实现了一个基于守护进程的分布式架构，它负责在同一台机器上运行的查询执行的所有方面。因此，它减少了使用MapReduce的延迟，这使Impala比Hive快。

- Impala与Kudu间的关系

  Kudu与Impala紧密集成，替代Impala+HDFS+Parquet组合。允许使用Impala的SQL语法从Kudu tablets插入、查询、更新和删除数据。此外，还可以用JDBC或ODBC，Impala作为代理连接Kudu进行数据操作。

- Impala与HBase间的关系

  Impala表默认使用存储在HDFS上的数据文件，便于全表扫描的批量加载和查询。但是，HBase可以提供对OLTP样式组织的数据的便捷高效查询。

6.16 IoTDB

6.16.1 IoTDB 基本原理

IoTDB（物联网数据库）是一体化收集、存储、管理与分析物联网时序数据的软件系统。 Apache IoTDB采用轻量式架构，具有高性能和丰富的功能。

IoTDB从存储上对时间序列进行排序，索引和chunk块存储，大大地提升时序数据的查询性能。通过Raft协议，来确保数据的一致性。针对时序场景，对存储数据进行预计算和存储，提升分析场景的性能。针对时序数据特征，进行强有力的数据编码和压缩能力，同时其自身的副本机制也保证了数据的安全，并与Apache Hadoop和Flink等进行了深度集成，可以满足工业物联网领域的海量数据存储、高速数据读取和复杂数据分析需求。

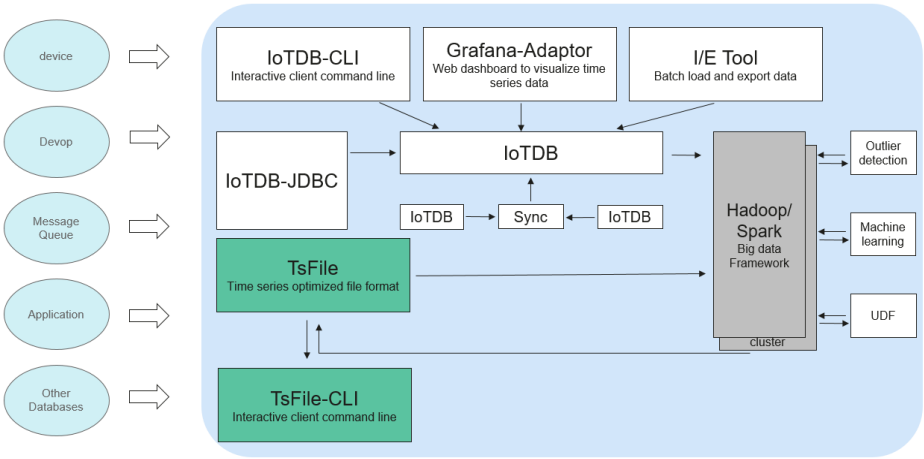
更多关于IoTDB组件操作指导，请参考**使用IoTDB**。

IoTDB 结构

IoTDB套件由若干个组件构成，共同形成数据收集、数据写入、数据存储、数据查询、数据可视化、数据分析等一系列功能。

图6-65展示了使用IoTDB套件的全部组件形成的整体应用架构，IoTDB特指其中的时间序列数据库组件。

图 6-65 IoTDB 结构

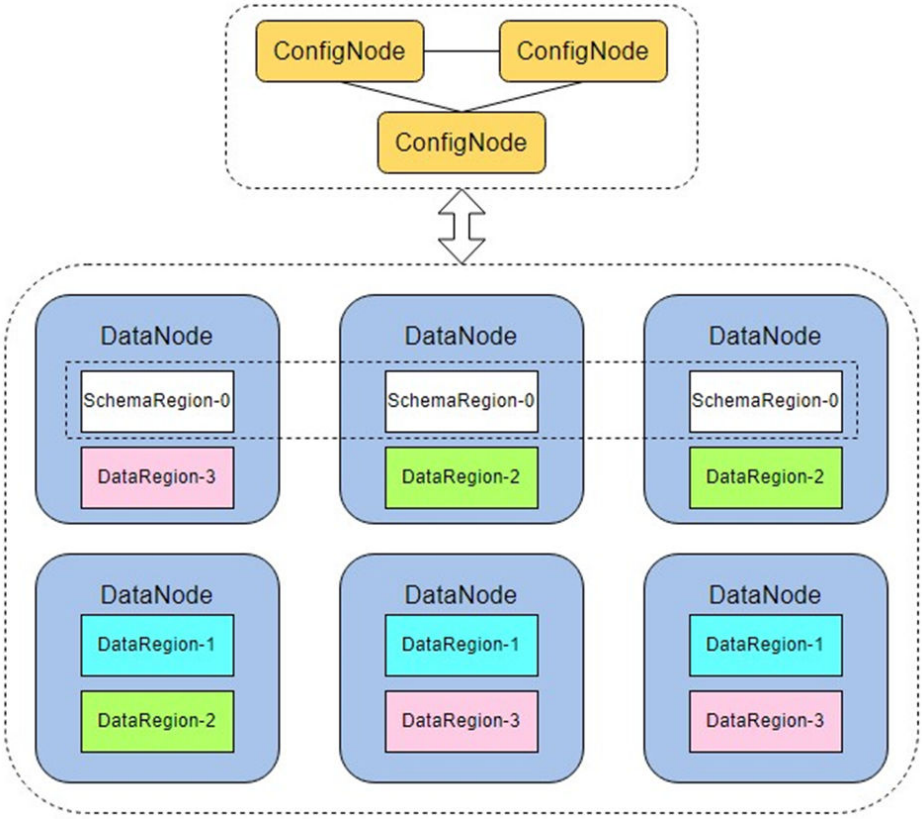


- 用户可以通过JDBC/Session将来自设备传感器上采集的时序数据和服务器负载、CPU内存等系统状态数据、消息队列中的时序数据、应用程序的时序数据或者其他数据库中的时序数据导入到本地或者远程的IoTDB中。用户还可以将上述数据直接写成本地（或位于HDFS上）的TsFile文件。
- 用户可以将TsFile文件写入到HDFS上，进而满足Hadoop、Flink等数据处理任务的访问。
- 对于写入到HDFS或者本地的TsFile文件，可以利用TsFile-Hadoop或TsFile-Flink连接器，允许Hadoop或Flink进行数据处理。
- 对于分析的结果，可以写回成TsFile文件。
- IoTDB和TsFile还提供了相应的客户端工具，满足用户以SQL形式、脚本形式和图形形式写入和查看数据的各种需求。

IoTDB服务包括IoTDBServer（DataNode）和ConfigNode两种角色。由于社区版角色名称DataNode和HDFS角色同名，因此将DataNode更名为IoTDBServer，如**图6-66**所示。

- ConfigNode：管理角色，负责DataNode数据分片，负载均衡等。
- IoTDBServer（DataNode）：存储角色，负责数据存储、查询和写入等功能。

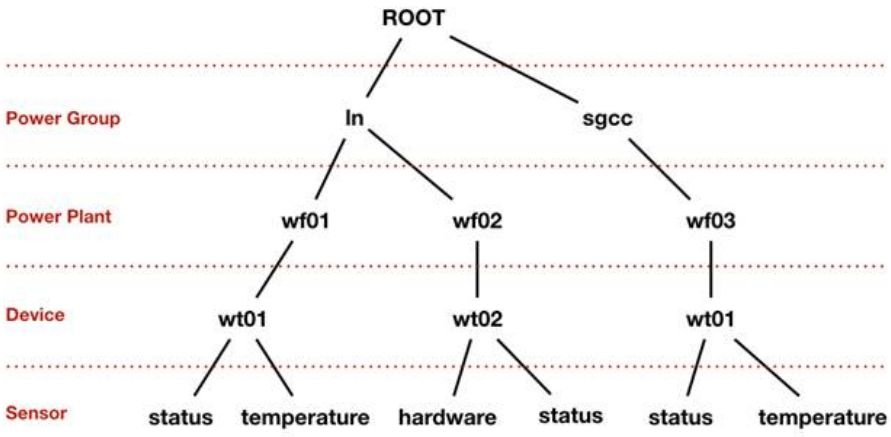
图 6-66 IoTDB 分布式架构



IoTDB 原理

根据属性层级、属性涵盖范围以及数据之间的从属关系，可将IoTDB数据模型表示为如图6-67所示的属性层级组织结构，即“电力集团层-电厂层-设备层-传感器层”。其中ROOT为根节点，传感器层的每一个节点为叶子节点。IoTDB的语法规定，ROOT节点到叶子节点的路径以“.”连接，以此完整路径命名IoTDB中的一个时间序列。例如，下图最左侧路径对应的时间序列名称为“ROOT.ln.wf01.wt01.status”。

图 6-67 IoTDB 数据模型



IoTDB 与其他组件的关系

IoTDB存储数据在本地，因此在存储上不依赖于其他任何组件。但是安全集群的环境上，IoTDB依赖于KrbServer组件来进行Kerberos认证。

6.16.2 IoTDB 开源增强特性

IoTDB 开源增强特性：可视化

- 可视化运维，包含安装、卸载、一键启动和停止、配置、客户端、监控、告警、健康检查、日志。
- 可视化权限管理，无需后台命令行操作，支持库表级别读写权限控制。
- 日志级别的可视化配置动态生效、可视化下载、可视化检索、审计日志等功能。

IoTDB 开源增强特性：安全加固

用户认证支持Kerberos、通道SSL加密，兼容社区方式。

IoTDB 开源增强特性：生态对接

在原生的能力上，增强集群版MQTT对接。

6.17 JobGateway

JobGateway组件提供REST API作业提交能力。

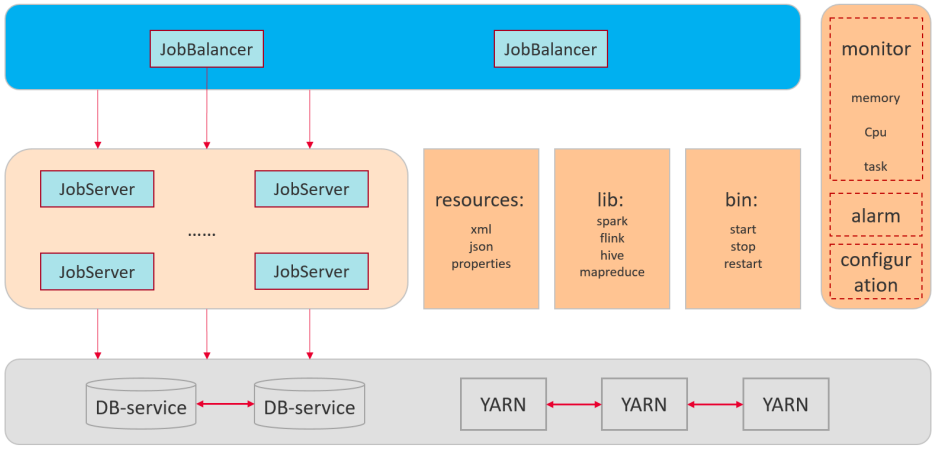
JobGateway作为大数据作业提交功能的网关组件，提供完全可控的企业级大数据作业提交服务，可轻松完成Spark、Flink、Hive等大数据作业任务提交功能。更多关于JobGateway组件操作指导，请参考**使用JobGateway**。

JobGateway 结构

JobGateway组件由JobServer实例以及JobBalancer实例组成。

- JobBalancer提供负载均衡能力。
- JobServer提供REST API提供作业提交能力。

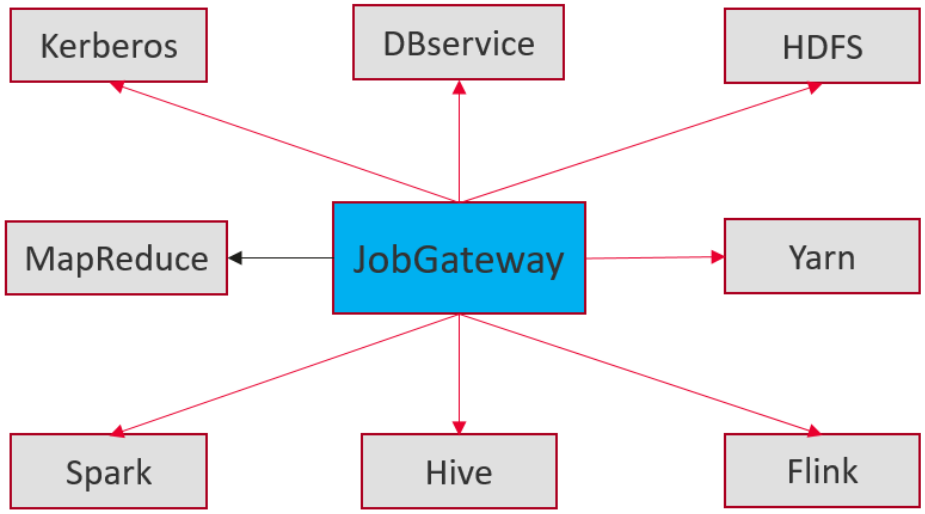
图 6-68 JobGateway 结构



JobGateway 与其他组件的关系

JobGateway是提供作业提交的REST API服务，提供Spark、Hive、MapReduce、Flink提交作业能力。

图 6-69 JobGateway 与其他组件的关系



6.18 Kafka

6.18.1 Kafka 基本原理

Kafka是一个分布式的、分区的、多副本的消息发布-订阅系统，它提供了类似于JMS的特性，但在设计上完全不同，它具有消息持久化、高吞吐、分布式、多客户端支持、实时等特性，适用于离线和在线的消息消费，如常规的消息收集、网站活性跟踪、聚合统计系统运营数据（监控数据）、日志收集等大量数据的互联网服务的数据收集场景。

更多关于Kafka组件操作指导，请参考**使用Kafka**。

Kafka 结构

生产者（Producer）将消息发布到Kafka主题（Topic）上，消费者（Consumer）订阅这些主题并消费这些消息。在Kafka集群上一个服务器称为一个Broker。对于每一个主题，Kafka集群保留一个用于缩放、并行化和容错性的分区（Partition）。每个分区是一个有序、不可变的消息序列，并不断追加到提交日志文件。分区的消息每个也被赋值一个称为偏移顺序（Offset）的序列化编号。

图 6-70 Kafka 结构

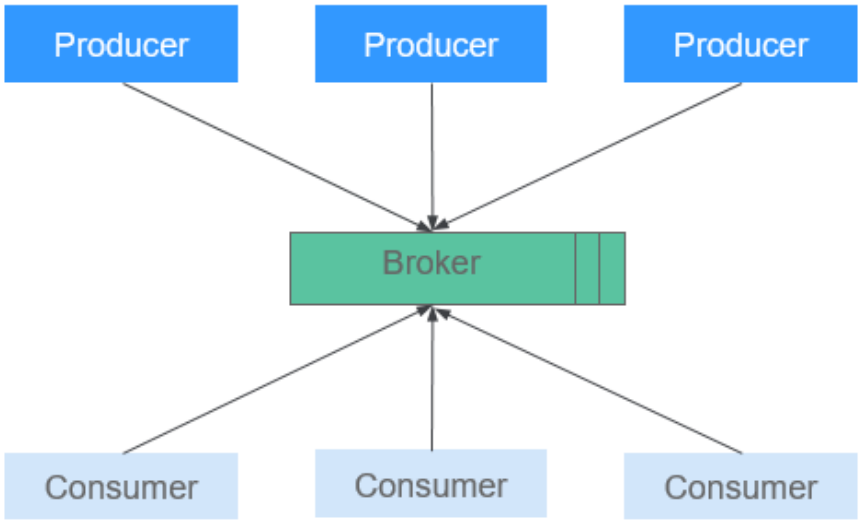
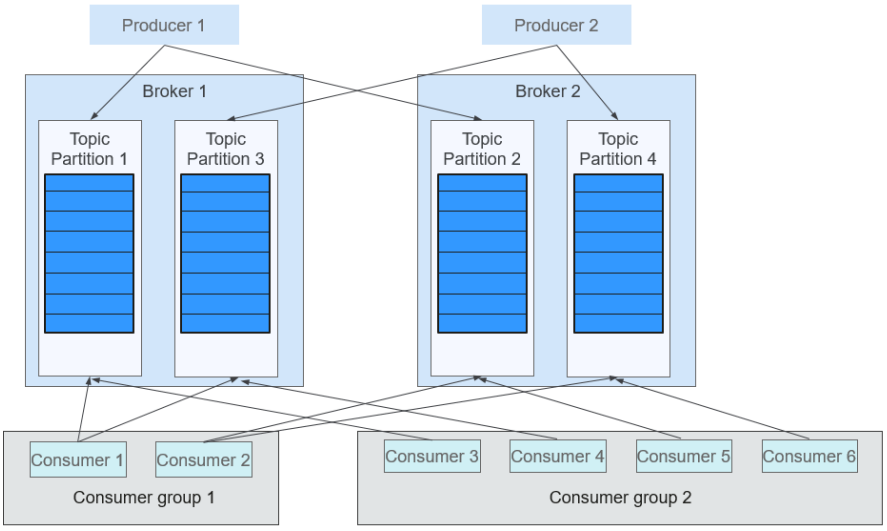


表 6-16 Kafka 结构图说明

名称	说明
Broker	在Kafka集群上一个服务器称为一个Broker。
Topic/主题	一个Topic就是一个类别或者一个可订阅的条目名称，也即一类消息。一个主题可以有多个分区，这些分区可以作为并行的一个单元。
Partition/分区	是一个有序的、不可变的消息序列，这个序列可以被连续地追加一个提交日志。在分区内的每条消息都有一个有序的ID号，这个ID号被称为偏移（Offset），这个偏移量可以唯一确定每条消息在分区内的位置。
Producer/生产者	向Kafka的主题发布消息。
Consumer/消费者	向Topic订阅，并且接收发布到这些Topic的消息。

各模块间关系如图6-71所示。

图 6-71 Kafka 模块间关系



消费者使用一个消费者组名称来标记自己，主题的每个消息被传递给每个订阅消费者组中的一个消费者。如果所有的消费者实例都属于同样的消费组，它们就以传统队列负载均衡方式工作。如上图中，Consumer1与Consumer2之间为负载均衡方式；Consumer3、Consumer4、Consumer5与Consumer6之间为负载均衡方式。如果消费者实例都属于不同的消费组，则消息会被广播给所有消费者。如上图中，Topic1中的消息，同时会广播到Consumer Group1与Consumer Group2中。

关于Kafka架构和详细原理介绍，请参见：**https://kafka.apache.org/24/documentation.html**。

Kafka 原理

- **消息可靠性**

  Kafka Broker收到消息后，会持久化到磁盘，同时，Topic的每个Partition有自己的Replica（备份），每个Replica分布在不同的Broker节点上，以保证当某一节点失效时，可以自动故障转移到可用消息节点。

- **高吞吐量**

  Kafka通过以下方式提供系统高吞吐量：

  - 数据磁盘持久化：消息不在内存中缓存，直接写入到磁盘，充分利用磁盘的顺序读写性能。
  - Zero-copy：减少IO操作步骤。
  - 数据批量发送：提高网络利用率。
  - Topic划分为多个Partition，提高并发度，可以由多个Producer、Consumer数目之间的关系并发来读、写消息。Producer根据用户指定的算法，将消息发送到指定的Partition。

- **消息订阅-通知机制**

  消费者对感兴趣的主题进行订阅，并采取pull的方式消费数据，使得消费者可以根据其消费能力自主地控制消息拉取速度，同时，可以根据自身情况自主选择消费模式，例如批量、重复消费，从尾端开始消费等；另外，需要消费者自己负责维护其自身消息的消费记录。

- **可扩展性**

  当在Kafka集群中可通过增加Broker节点以提供更大容量时。新增的Broker会向ZooKeeper注册，而Producer及Consumer会及时从ZooKeeper感知到这些变化，并做出调整。

Kafka 开源特性

- 可靠性

  提供At-Least Once，At-Most Once，Exactly Once消息可靠传递。消息被处理的状态是在Consumer端维护，需要结合应用层实现Exactly Once。

- 高吞吐

  同时为发布和订阅提供高吞吐量。

- 持久化

  将消息持久化到磁盘，因此可用于批量消费以及实时应用程序。通过将数据持久化到硬盘以及replication的方式防止数据丢失。

- 分布式

  分布式系统，易于向外扩展。每个集群支持部署多个Producer、Broker和Consumer，从而形成分布式的集群，无需停机即可扩展系统。

Kafka UI

Kafka UI提供Kafka Web服务，通过界面展示Kafka集群中Broker、Topic、Partition、Consumer等功能模块的基本信息，同时提供Kafka服务常用命令的界面操作入口。该功能作为Kafka Manager替代，提供符合安全规范的Kafka Web服务。

通过Kafka UI可以进行以下操作：

- 支持界面检查集群状态（主题，消费者，偏移量，分区，副本，节点）
- 支持界面执行集群内分区重新分配
- 支持界面选择配置创建主题
- 支持界面删除主题（Kafka服务设置了参数“delete.topic.enable = true”）
- 支持为已有主题增加分区
- 支持更新现有主题的配置
- 可以为分区级别和主题级别度量标准启用JMX查询

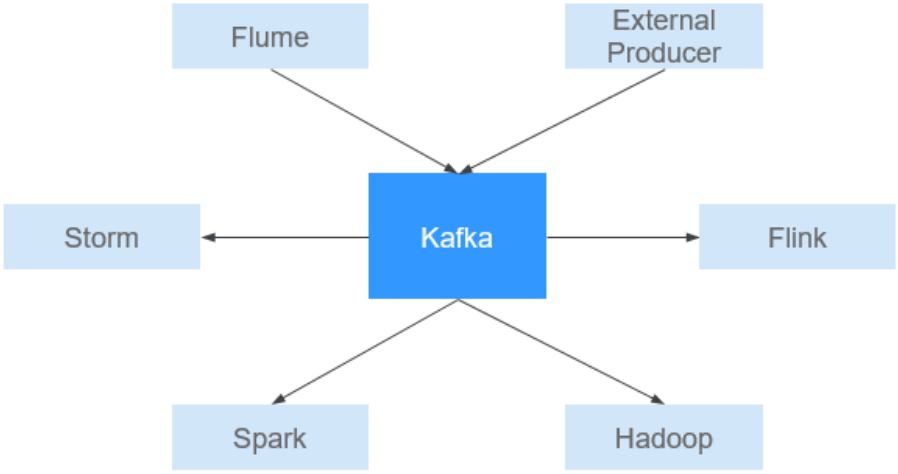
6.18.2 Kafka 与其他组件的关系

Kafka作为一个消息发布-订阅系统，为整个大数据平台多个子系统之间数据的传递提供了高速数据流转方式。

Kafka可以实时接收来自外部的消息，并提供给在线以及离线业务进行处理。

Kafka与其他组件的具体的关系如下图所示：

图 6-72 与其他组件关系



6.18.3 Kafka 开源增强特性

Kafka 开源增强特性

- 支持监控如下Topic级别的指标：
  - Topic输入的字节流量
  - Topic输出的字节流量
  - Topic拒绝的字节流量
  - Topic每秒失败的fetch请求数
  - Topic每秒失败的Produce请求数
  - Topic每秒输入的消息条数
  - Topic每秒的fetch请求数
  - Topic每秒的produce请求数
- 支持查询Broker ID与节点IP的对应关系。在Linux客户端下，使用**kafka-broker-info.sh**查询Broker ID与节点IP的对应关系。

6.19 KafkaManager

KafkaManager是Apache Kafka的管理工具，提供Kafka集群界面化的Metric监控和集群管理。

通过KafkaManager进行以下操作：

- 支持管理多个Kafka集群
- 支持界面检查集群状态（主题，消费者，偏移量，分区，副本，节点）
- 支持界面执行副本的leader选举
- 使用选择生成分区分配以选择要使用的分区方案
- 支持界面执行分区重新分配（基于生成的分区方案）

- 支持界面选择配置创建主题（支持多种Kafka版本集群）
- 支持界面删除主题（仅0.8.2版本并设置参数“delete.topic.enable = true”的集群支持）
- 支持批量生成多个主题的分区分配，并可选择要使用的分区方案
- 支持批量运行重新分配多个主题的分区
- 支持为已有主题增加分区
- 支持更新现有主题的配置
- 可以为分区级别和主题级别度量标准启用JMX查询
- 可以过滤掉zookeeper中没有ids / owner /&offsets /目录的使用者。

6.20 KrbServer 及 LdapServer

6.20.1 KrbServer 及 LdapServer 基本原理

KrbServer 及 LdapServer 简介

为了管理集群中数据与资源的访问控制权限，推荐安装安全模式集群。在安全模式下，客户端应用程序在访问集群中的任意资源之前均需要通过身份认证，建立安全会话链接。MRS通过KrbServer为所有组件提供Kerberos认证功能，实现了可靠的认证机制。

LdapServer支持轻量目录访问协议（Lightweight Directory Access Protocol，简称为LDAP），为Kerberos认证提供用户和用户组数据保存能力。

KrbServer 及 LdapServer 结构

用户登录时安全认证功能主要依赖于Kerberos和LDAP。

图 6-73 安全认证场景架构

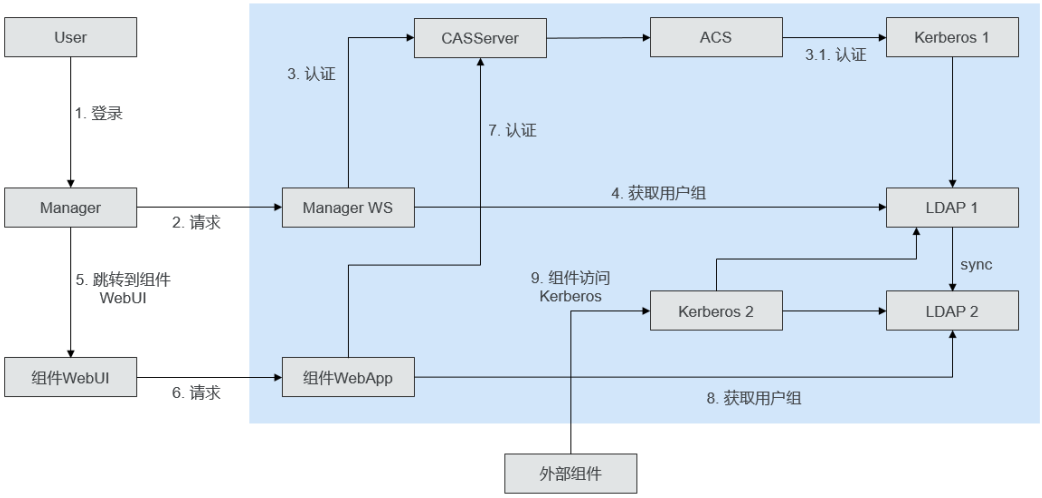


图6-73可分为三类场景：

- 登录Manager WebUI
  认证架构包含步骤1、2、3、4
- 登录组件Web UI
  认证架构包含步骤5、6、7、8
- 组件间访问
  认证架构为步骤9

表 6-17 关键模块解释

名称	含义
Manager	集群Manager
Manager WS	WebBrowser
Kerberos1	部署在Manager中的KrbServer（管理平面）服务，即OMS Kerberos
Kerberos2	部署在集群中的KrbServer（业务平面）服务
LDAP1	部署在Manager中的LdapServer（管理平面）服务，即OMS LDAP
LDAP2	部署在集群中的LdapServer（业务平面）服务

Kerberos1访问LDAP数据：以负载均衡方式访问主备LDAP1两个实例和双备LDAP2两个实例。只能在主LDAP1主实例上进行数据的写操作，可以在LDAP1或者LDAP2上进行数据的读操作。

Kerberos2访问LDAP数据：读操作可以访问LDAP1和LDAP2，数据的写操作只能在主LDAP1实例进行。

KrbServer 及 LdapServer 原理

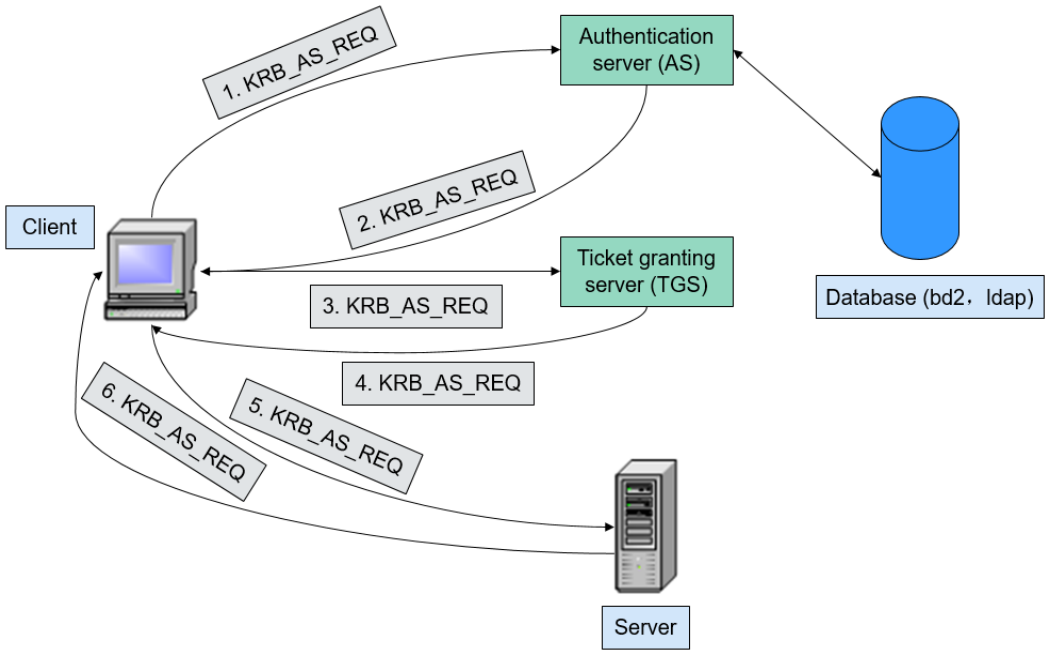
Kerberos认证

Kerberos作为安全认证的概念，该系统设计上采用客户端/服务器结构与DES、AES等加密技术，并且能够进行相互认证，即客户端和服务器端均可对对方进行身份认证。可以用于防止窃听、防止replay攻击、保护数据完整性等场景，是一种应用对称密钥体制进行密钥管理的系统。

Kerberos认证协议，主要包含三个角色：

- Client：客户端
- Server：客户端需要请求的服务端
- KDC（Key Distribution Center）: 密钥分发中心，包括AS和TGS两部分。
  - AS（Authentication Server）: 认证服务器，用于验证客户端账号密码信息，并生成TGT（Ticket Granting Ticket） 票据授权票据。
  - TGS（Ticket Granting Server）: 票据授权服务器，用于通过TGT生成访问服务的服务票据ST。

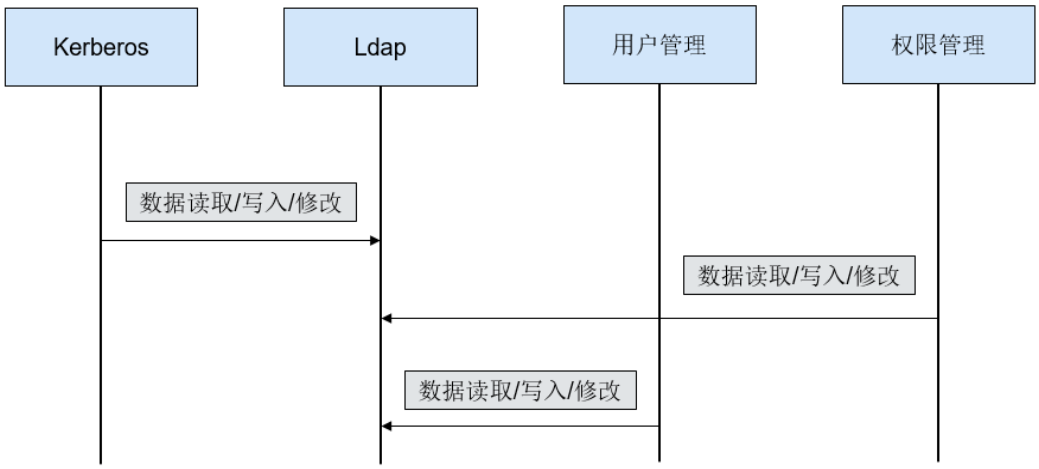
图 6-74 认证流程图



LDAP数据读写

LDAP作为用户数据存储中心，存储了集群内用户的信息，包含密码，附属信息等。用户操作用户数据或进行Kerberos认证需要访问LDAP。

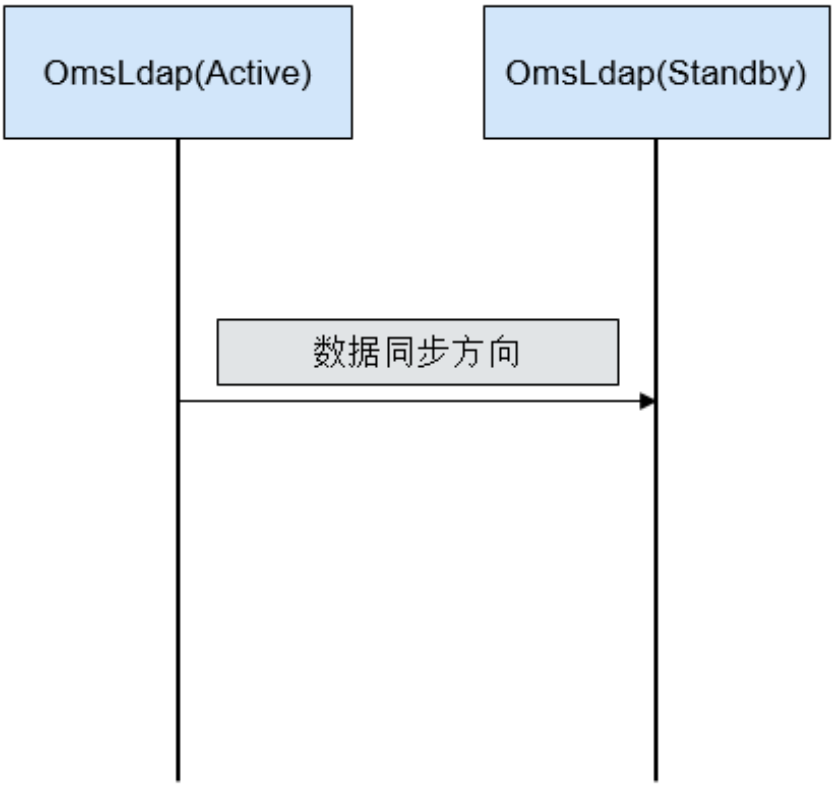
图 6-75 数据修改过程



LDAP数据同步

- 安装集群前OMS LDAP数据同步

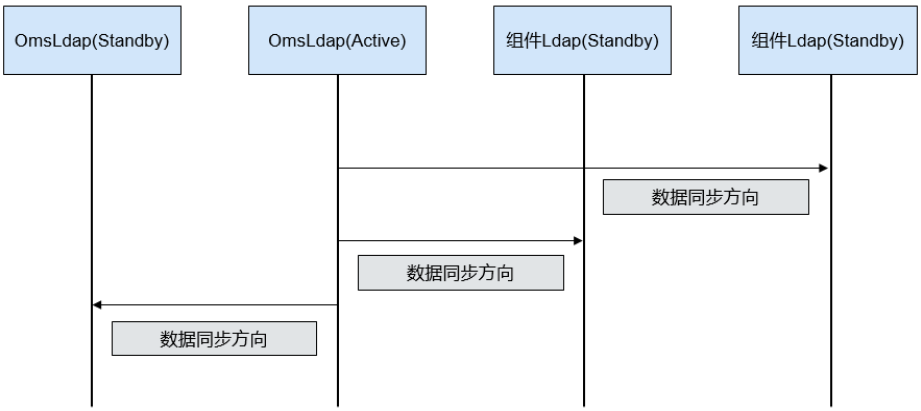
图 6-76 OMS LDAP 数据同步



安装集群前数据同步方向：主OMS LDAP同步到备OMS LDAP。

- 安装集群后LDAP数据同步

图 6-77 LDAP 数据同步



安装集群后数据同步方向：主OMS LDAP同步到备OMS LDAP、备组件LDAP和备组件LDAP。

6.20.2 KrbServer 及 LdapServer 开源增强特性

集群内服务认证

在使用安全模式的MRS集群中，任意服务间的相互访问基于Kerberos安全架构方案。集群内某个服务（例如HDFS）在启动准备阶段的时候，会首先在Kerberos中获取该服务对应的服务名称sessionkey（即keytab，用于应用程序进行身份认证）。其他任意服务（例如YARN）需要访问HDFS并在HDFS中执行增、删、改、查数据的操作时，必须获取对应的TGT和ST，用于本次安全访问的认证。

应用开发认证

MRS各组件提供了应用开发接口，用于用户或者上层业务产品集群使用。在应用开发过程中，安全模式的集群提供了特定的应用开发认证接口，用于应用程序的安全认证与访问。例如hadoop-common api提供的UserGroupInformation类，该类提供了多个安全认证API接口：

- setConfiguration()主要是获取对应的配置，设置全局变量等参数。
- loginUserFromKeytab()获取TGT接口。

跨系统互信特性

MRS提供两个Manager之间的互信功能，用于实现系统之间的数据读、写等操作。

6.21 Kudu

Kudu是专为Apache Hadoop平台开发的列式存储管理器，具有Hadoop生态系统应用程序的共同技术特性：在通用的商用硬件上运行，可水平扩展，提供高可用性。

Kudu的设计具有以下优点：

- 能够快速处理OLAP工作负载
- 支持与MapReduce，Spark和其他Hadoop生态系统组件集成
- 与Apache Impala的紧密集成，使其成为将HDFS与Apache Parquet结合使用的更好选择
- 提供强大而灵活的一致性模型，允许您根据每个请求选择一致性要求，包括用于严格可序列化的一致性的选项
- 提供同时运行顺序读写和随机读写的良好性能
- 易于管理
- 高可用性。Master和TServer采用raft算法，该算法可确保只要副本总数的一半以上可用，tablet就可以进行读写操作。例如，如果3个副本中有2个副本或5个副本中有3个副本可用，则tablet可用。即使主tablet出现故障，也可以通过只读的副tablet提供读取服务
- 支持结构化数据模型

通过结合所有以上属性，Kudu的目标是支持在当前Hadoop存储技术上难以实现或无法实现的应用。

Kudu的应用场景有：

- 需要最终用户立即使用新到达数据的报告型应用

- 同时支持大量历史数据查询和细粒度查询的时序应用
- 使用预测模型并基于所有历史数据定期刷新预测模型来做出实时决策的应用

更多关于Kudu组件操作指导，请参考**使用Kudu**。

Kudu 与其他组件的关系

- Kudu与HBase的关系：

  Kudu的设计参考了HBase的结构，能够实现HBase擅长的快速随机读写、更新的功能。

  二者主要差别在于：

  - Kudu不依赖Zookeeper，通过自身实现Raft来保证一致性。
  - Kudu持久化数据不依赖HDFS，TServer实现数据的强一致性和可靠性。

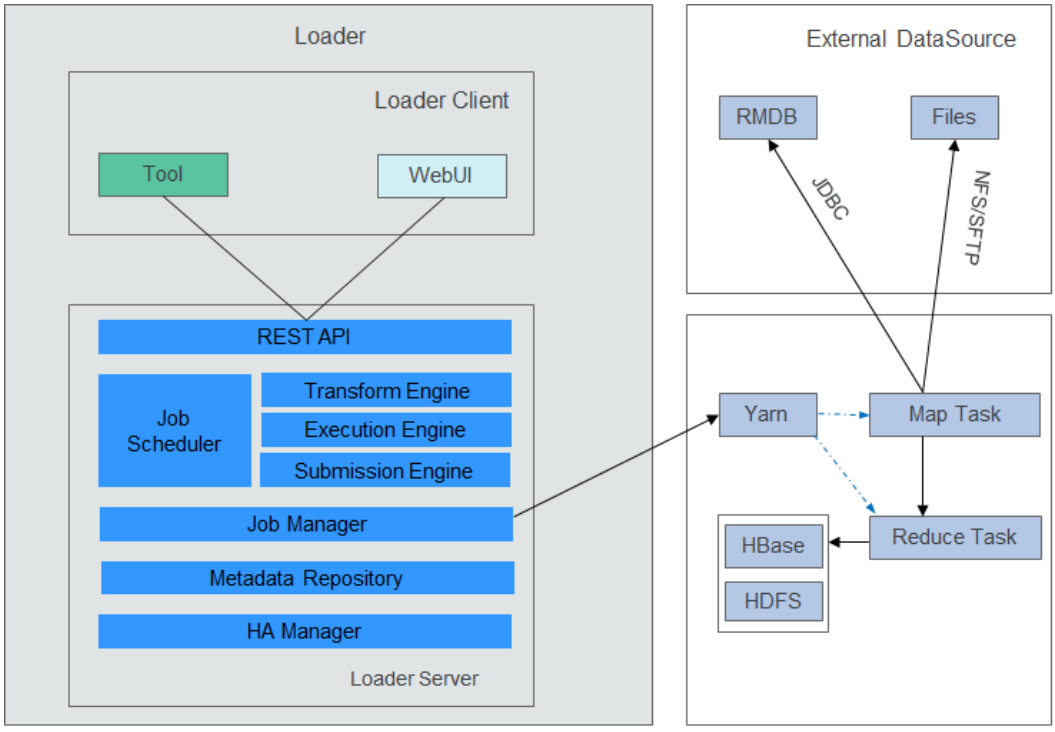
6.22 Loader

6.22.1 Loader 基本原理

Loader是在开源Sqoop组件的基础上进行了一些扩展，实现MRS与关系型数据库、文件系统之间交换“数据”、“文件”，同时也可以将数据从关系型数据库或者文件服务器导入到HDFS/HBase中，或者反过来从HDFS/HBase导出到关系型数据库或者文件服务器中。

Loader模型主要由Loader Client和Loader Server组成，如**图6-78**所示。

图 6-78 Loader 模型



上图中各部分的功能说明如**表6-18**所示。

表 6-18 Loader 模型组成

名称	描述
Loader Client	Loader的客户端，包括WebUI和CLI版本两种交互界面。
Loader Server	Loader的服务端，主要功能包括：处理客户端操作请求、管理连接器和元数据、提交MapReduce作业和监控MapReduce作业状态等。
REST API	实现RESTful（HTTP + JSON）接口，处理来自客户端的操作请求。
Job Scheduler	简单的作业调度模块，支持周期性的执行Loader作业。
Transform Engine	数据转换处理引擎，支持字段合并、字符串剪切、字符串反序等。
Execution Engine	Loader作业执行引擎，支持以MapReduce方式执行Loader作业。
Submission Engine	Loader作业提交引擎，支持将作业提交给MapReduce执行。
Job Manager	管理Loader作业，包括创建作业、查询作业、更新作业、删除作业、激活作业、去激活作业、启动作业、停止作业。
Metadata Repository	元数据仓库，存储和管理Loader的连接器、转换步骤、作业等数据。
HA Manager	管理Loader Server进程的主备状态，Loader Server包含2个节点，以主备方式部署。

Loader通过MapReduce作业实现并行的导入或者导出作业任务，不同类型的导入导出作业可能只包含Map阶段或者同时Map和Reduce阶段。

Loader同时利用MapReduce实现容错，在作业任务执行失败时，可以重新调度。

- **数据导入到HBase**

  在MapReduce作业的Map阶段中从外部数据源抽取数据。

  在MapReduce作业的Reduce阶段中，按Region的个数启动同样个数的Reduce Task，Reduce Task从Map接收数据，然后按Region生成HFile，存放在HDFS临时目录中。

  在MapReduce作业的提交阶段，将HFile从临时目录迁移到HBase目录中。

- **数据导入HDFS**

  在MapReduce作业的Map阶段中从外部数据源抽取数据，并将数据输出到HDFS临时目录下（以“*输出目录*-ldtmp”命名）。

  在MapReduce作业的提交阶段，将文件从临时目录迁移到输出目录中。

- **数据导出到关系型数据库**

  在MapReduce作业的Map阶段，从HDFS或者HBase中抽取数据，然后将数据通过JDBC接口插入到临时表（Staging Table）中。

  在MapReduce作业的提交阶段，将数据从临时表迁移到正式表中。

- **数据导出到文件系统**

  在MapReduce作业的Map阶段，从HDFS或者HBase中抽取数据，然后将数据写入到文件服务器临时目录中。

  在MapReduce作业的提交阶段，将文件从临时目录迁移到正式目录。

Loader的架构和详细原理介绍，请参见：**https://sqoop.apache.org/docs/1.99.3/index.html**。

更多关于Loader组件操作指导，请参考**使用Loader**。

6.22.2 Loader 与其他组件的关系

与Loader有交互关系的组件有HDFS、HBase、Hive、Yarn、Mapreduce和ZooKeeper等。

Loader作为客户端使用这些组件的某些功能，如存储数据到HDFS和HBase，从HDFS和HBase表读数据，同时Loader本身也是一个Mapreduce客户端程序，完成一些数据导入导出任务。

Loader通过MapReduce作业实现并行的导入或者导出作业任务，不同类型的导入导出作业可能只包含Map阶段或者同时Map和Reduce阶段。

6.22.3 Loader 开源增强特性

Loader 开源增强特性：数据导入导出

Loader是在开源Sqoop组件的基础上进行了一些扩展，除了包含Sqoop开源组件本身已有的功能外，还开发了如下的增强特性：

- 提供数据转化功能
- 支持图形化配置转换步骤
- 支持从SFTP/FTP服务器导入数据到HDFS/OBS
- 支持从SFTP/FTP服务器导入数据到HBase表
- 支持从SFTP/FTP服务器导入数据到Phoenix表
- 支持从SFTP/FTP服务器导入数据到Hive表
- 支持从HDFS/OBS导出数据到SFTP服务器
- 支持从HBase表导出数据到SFTP服务器
- 支持从Phoenix表导出数据到SFTP服务器
- 支持从关系型数据库导入数据到HBase表
- 支持从关系型数据库导入数据到Phoenix表
- 支持从关系型数据库导入数据到Hive表
- 支持从HBase表导出数据到关系型数据库
- 支持从Phoenix表导出数据到关系型数据库
- 支持从Oracle分区表导入数据到HDFS/OBS
- 支持从Oracle分区表导入数据到HBase表
- 支持从Oracle分区表导入数据到Phoenix表
- 支持从Oracle分区表导入数据到Hive表

- 支持从HDFS/OBS导出数据到Oracle分区表
- 支持从HBase导出数据到Oracle分区表
- 支持从Phoenix表导出数据到Oracle分区表
- 在同一个集群内，支持从HDFS导数据到HBase、Phoenix表和Hive表
- 在同一个集群内，支持从HBase和Phoenix表导数据到HDFS/OBS
- 导入数据到HBase和Phoenix表时支持使用bulkload和put list两种方式
- 支持从SFTP/FTP导入所有类型的文件到HDFS，开源只支持导入文本文件
- 支持从HDFS/OBS导出所有类型的文件到SFTP，开源只支持导出文本文件和sequence格式文件
- 导入（导出）文件时，支持对文件进行转换编码格式，支持的编码格式为jdk支持的所有格式
- 导入（导出）文件时，支持保持原来文件的目录结构和文件名不变
- 导入（导出）文件时，支持对文件进行合并，如输入文件为海量个文件，可以合并为*n*个文件（*n*值可配）
- 导入（导出）文件时，可以对文件进行过滤，过滤规则同时支持通配符和正则表达式
- 支持批量导入/导出ETL任务
- 支持ETL任务分页查询、关键字查询和分组管理
- 对外部组件提供浮动IP

6.23 Manager

6.23.1 Manager 基本原理

Manager 功能

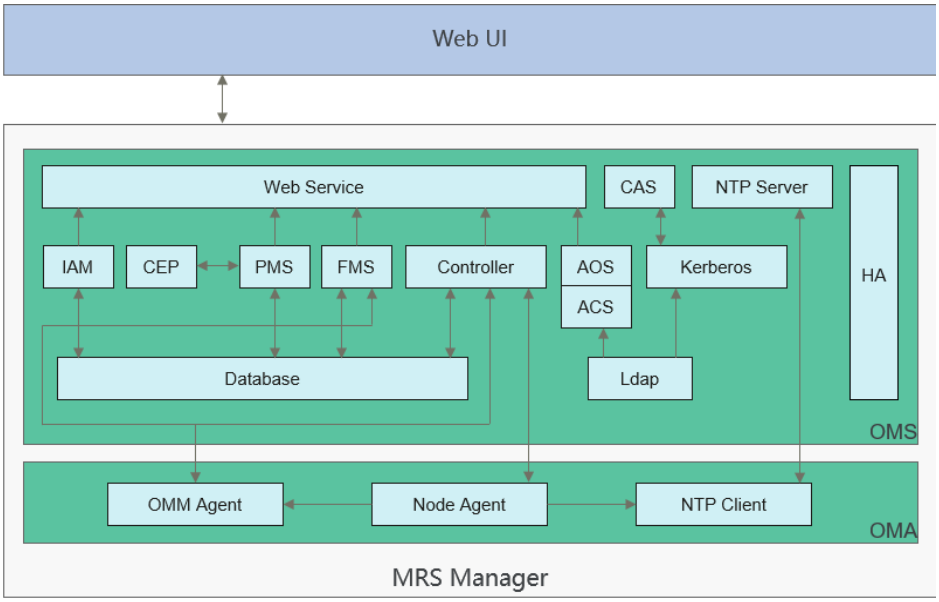
Manager是MRS的运维管理系统，为部署在集群内的服务提供统一的集群管理能力。

Manager支持大规模集群的性能监控、告警、用户管理、权限管理、审计、服务管理、健康检查、日志采集等功能。

Manager 结构

Manager的整体逻辑架构如**图6-79**所示。

图 6-79 Manager 逻辑架构



Manager由OMS和OMA组成：

- OMS：操作维护系统的管理节点，OMS一般有两个，互为主备。
- OMA：操作维护系统中的被管理节点，一般有多个。

图6-79中各模块的说明如表6-19所示：

表 6-19 业务模块说明

模块名称	描述
Web Service	是一个部署在Tomcat下的Web服务，提供Manager的https接口，用于通过浏览器访问Manager。同时还提供基于Syslog和SNMP协议的北向接入能力。
OMS	操作维护系统的管理节点，OMS节点一般有两个，互为主备。
OMA	操作维护系统中的被管理节点，一般有多个。
Controller	Controller是Manager的控制中心，负责汇聚来自集群中所有节点的信息，统一向MRS集群管理员展示，以及负责接收来自MRS集群管理员的操作指令，并且依据操作指令所影响的范围，向集群的所有相关节点同步信息。 Manager的控制进程，负责各种管理动作的执行： 1. Web Service将各种管理动作（安装、启停服务、修改配置等）下发到Controller。 2. Controller将命令分解，分解后将动作下发到每一个Node Agent。例如启动一个服务，会涉及多个角色和实例。 3. Controller负责监控每一个动作的执行情况。

模块名称	描述
Node Agent	Node Agent存在于每一个集群节点，是Manager在单个节点的使能器。 ● Node Agent代表本节点上部署的所有组件与Controller交互，实现整个集群多点到单点的汇聚。 ● Node Agent是Controller对部署在该节点上组件做一切操作的使能器，其代表着Controller的功能。 Node Agent每隔3秒向Controller发送心跳信息，不支持配置时间间隔。
IAM	负责记录审计日志。在Manager的UI上每一个非查询类操作，都有对应的审计日志。
PMS	性能监控模块，搜集每一个OMA上的性能监控数据并提供查询。
CEP	汇聚功能模块。比如将所有OMA上的磁盘已用空间汇总成一个性能指标。
FMS	告警模块，收集每一个OMA上的告警并提供查询。
OMM Agent	OMA上面性能监控和告警的Agent，负责收集该Agent Node上的性能监控数据和告警数据。
CAS	统一认证中心，登录Web Service时需要在CAS进行登录认证，浏览器通过URL自动跳转访问CAS。
AOS	权限管理模块，管理用户和用户组的权限。
ACS	用户和用户组管理模块，管理用户及用户归属的用户组。
Kerberos	在OMS与集群中各部署一个。 ● OMS Kerberos提供单点登录及Controller与Node Agent间认证的功能。 ● 集群中Kerberos提供组件用户安全认证功能，其服务名称为KrbServer，包含两种角色实例： – KerberosServer：认证服务器，为MRS提供安全认证使用。 – KerberosAdmin：管理Kerberos用户的进程。
Ldap	在OMS与集群中各部署一个。 ● OMS Ldap为用户认证提供数据存储。 ● 集群中的Ldap作为OMS Ldap的备份，其服务名称为LdapServer，角色实例为SlapdServer。
Database	Manager的数据库，负责存储日志、告警等信息。
HA	高可用性管理模块，主备OMS通过HA进行主备管理。
NTP Server NTP Client	负责同步集群内各节点的系统时钟。

6.23.2 Manager 关键特性

Manager 关键特性：统一监控告警

Manager提供可视化、便捷的监控告警功能。用户可以快速获取集群关键性能指标，并评测集群健康状态，同时提供性能指标的定制化显示功能及指标转换告警方法。Manager可监控所有组件的运行情况，并在故障时实时上报告警。通过界面的联机帮助，用户可以查看性能指标和告警恢复的详细方法，进行快速排障。

Manager 关键特性：统一用户权限管理

Manager提供系统中各组件的权限集中管理功能。

Manager引入角色的概念，采用RBAC的方式对系统进行权限管理，集中呈现和管理系统中各组件零散的权限功能，并且将各个组件的权限以权限集合（即角色）的形式组织，形成统一的系统权限概念。这样一方面对普通用户屏蔽了内部的权限管理细节，另一方面对MRS集群管理员简化了权限管理的操作方法，提升了权限管理的易用性和用户体验。

Manager 关键特性：单点登录

提供Manager WebUI与组件WebUI之间的单点登录，以及MRS与第三方系统集成时的单点登录。

此功能统一了Manager系统用户和组件用户的管理及认证。整个系统使用LDAP管理用户，使用Kerberos进行认证，并在OMS和组件间各使用一套Kerberos和LDAP的管理机制，通过CAS实现单点登录（包括单点登录和单点登出）。用户只需要登录一次，即可在Manager WebUI和组件Web UI之间，甚至第三方系统之间进行任务跳转操作，无需切换用户重新登录。

说明

- 出于安全考虑，CAS Server只能保留用户使用的TGT（ticket-granting ticket）20分钟。
- 如用户20分钟内不对页面（包括Manager和组件WebUI）进行操作，页面自动锁定。

Manager 关键特性：自动健康检查与巡检

Manager为用户提供界面化的系统运行环境自动检查服务，帮助用户实现一键式系统运行健康度巡检和审计，保障系统的正常运行，降低系统运维成本。用户查看检查结果后，还可导出检查报告用于存档及问题分析。

Manager 关键特性：租户管理

Manager引入了多租户的概念，集群拥有的CPU、内存和磁盘等资源，可以整合规划为一个集合体，这个集合体就是租户。多个不同的租户统称多租户。

多租户功能支持层级式的租户模型，支持动态的添加和删除租户，实现资源的隔离，可以对租户的计算资源和存储资源进行动态配置和管理。

- 计算资源指租户Yarn任务队列资源，可以修改任务队列的配额，并查看任务队列的使用状态和使用统计。
- 存储资源目前支持HDFS存储，可以添加删除租户HDFS存储目录，设置目录的文件数量配额和存储空间配额。

Manager作为MRS的统一租户管理平台，用户可以在界面上根据业务需要，在集群中创建租户、管理租户。

- 创建租户时将自动创建租户对应的角色、计算资源和存储资源。默认情况下，新的计算资源和存储资源的全部权限将分配给租户的角色。
- 修改租户的计算资源或存储资源，对应的角色关联权限将自动更新。

Manager还提供了多实例的功能，使用户在资源控制和业务隔离的场景中可以独立使用HBase、Hive和Spark组件。多实例功能默认关闭，可以选择手动启用。

Manager 关键特性：多语言支持

Manager增加了对多语言的支持，系统自动根据浏览器的语言偏好设置，显示中文或者英文。当浏览器首选语言是中文时，Manager显示中文界面；当浏览器首选语言不是中文时，Manager显示英文界面。用户也可以根据语言偏好，在界面左下角一键切换中英文界面（仅MRS 3.x及后续版本支持一键切换中英文界面）。

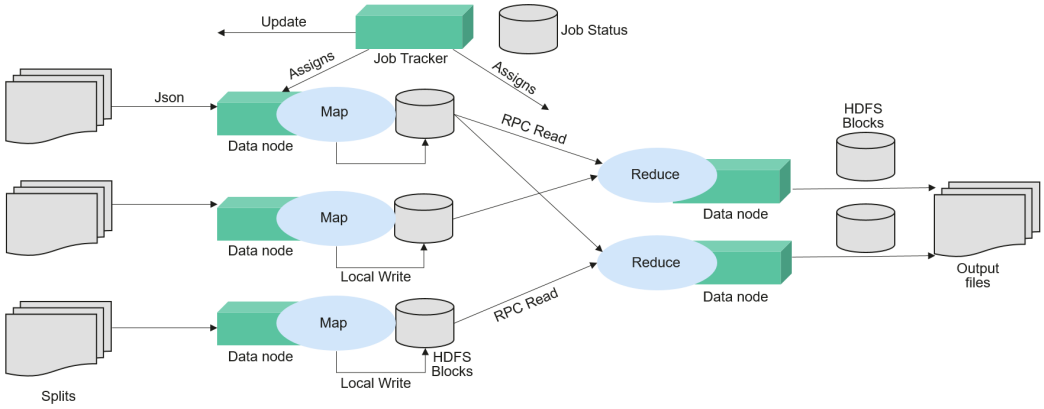
6.24 MapReduce

6.24.1 MapReduce 基本原理

MapReduce是Hadoop的核心，是Google提出的一个软件架构，用于大规模数据集（大于1TB）的并行运算。概念“Map（映射）”和“Reduce（化简）”及其主要思想，均取自于函数式编程语言及矢量编程语言。

当前的软件实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（化简）函数，用来保证所有映射的键值对共享相同的键组。

图 6-80 分布式批处理引擎



MapReduce是用于并行处理大数据集的软件框架。MapReduce的根源是函数性编程中的Map和Reduce函数。Map函数接受一组数据并将其转换为一个键/值对列表，输入域中的每个元素对应一个键/值对。Reduce函数接受Map函数生成的列表，然后根据它们的键缩小键/值对列表。MapReduce起到了将大事务分散到不同设备处理的能力，这样原来必须用单台较强服务器才能运行的任务，在分布式环境下也能完成。

更多信息，请参阅**MapReduce教程**。

更多关于MapReduce组件操作指导，请参考**使用MapReduce**。

说明

如需使用MapReduce，请确保MRS集群内已安装Hadoop服务。

MapReduce 结构

MapReduce通过实现YARN的Client和ApplicationMaster接口集成到YARN中，利用YARN申请计算所需资源。

6.24.2 MapReduce 与其他组件的关系

MapReduce 和 HDFS 的关系

- HDFS是Hadoop分布式文件系统，具有高容错和高吞吐量的特性，可以部署在价格低廉的硬件上，存储应用程序的数据，适合有超大数据集的应用程序。
- MapReduce是一种编程模型，用于大数据集（大于1TB）的并行运算。在MapReduce程序中计算的数据可以来自多个数据源，如Local FileSystem、HDFS、数据库等。最常用的是HDFS，利用HDFS的高吞吐性能读取大规模的数据进行计算，同时在计算完成后，也可以将数据存储到HDFS。

MapReduce 和 YARN 的关系

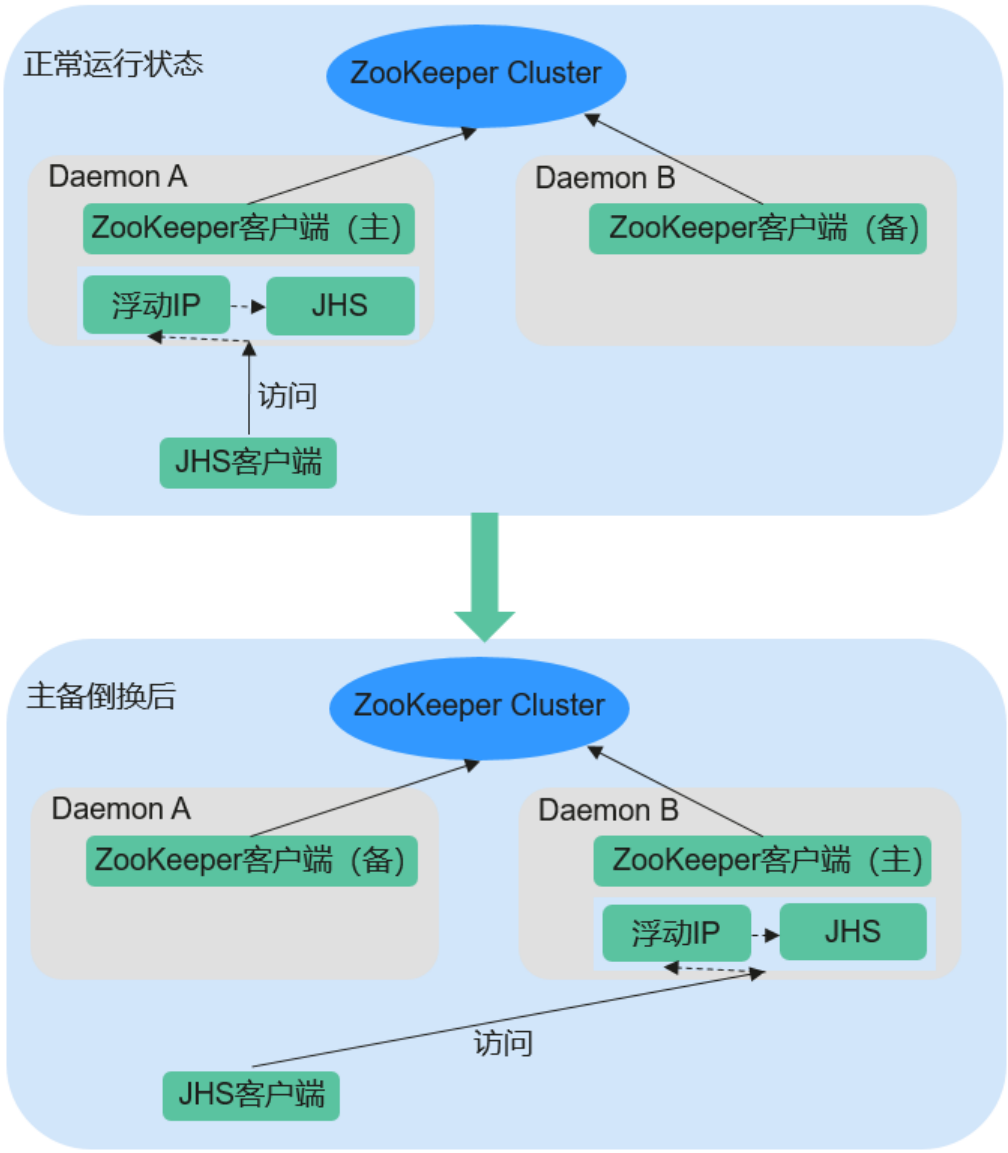
MapReduce是运行在YARN之上的一个批处理计算框架。MRv1是Hadoop 1.0中的MapReduce实现，它由编程模型（新旧编程接口）、运行时环境（由JobTracker和TaskTracker组成）和数据处理引擎（MapTask和ReduceTask）三部分组成。该框架在扩展性、容错性（JobTracker单点）和多框架支持（仅支持MapReduce一种计算框架）等方面存在不足。MRv2是Hadoop 2.0中的MapReduce实现，它在源码级重用了MRv1的编程模型和数据处理引擎实现，但运行时环境由YARN的ResourceManager和ApplicationMaster组成。其中ResourceManager是一个全新的资源管理系统，而ApplicationMaster则负责MapReduce作业的数据切分、任务划分、资源申请和任务调度与容错等工作。

6.24.3 MapReduce 开源增强特性

MapReduce 开源增强特性：JobHistoryServer HA 特性

JobHistoryServer（JHS）是用于查看MapReduce历史任务信息的服务器，当前开源JHS只支持单实例服务。JobHistoryServer HA能够解决JHS单点故障时，应用访问MapReduce接口无效，导致整体应用执行失败的场景，从而大大提升MapReduce服务的高可用性。

图 6-81 JobHistoryServer HA 主备倒换的状态转移过程



JobHistoryServer高可用性

- 采用ZooKeeper实现主备选举和倒换。
- JobHistoryServer使用浮动IP对外提供服务。
- 兼容JHS单实例，也支持HA双实例。
- 同一时刻，只有一个节点启动JHS进程，防止多个JHS操作同一文件冲突。
- 支持扩容减容、实例迁移、升级、健康检查等。

MapReduce 开源增强特性：特定场景优化 MapReduce 的 Merge/Sort 流程提升 MapReduce 性能

下图展示了MapReduce任务的工作流程。

图 6-82 MapReduce 作业

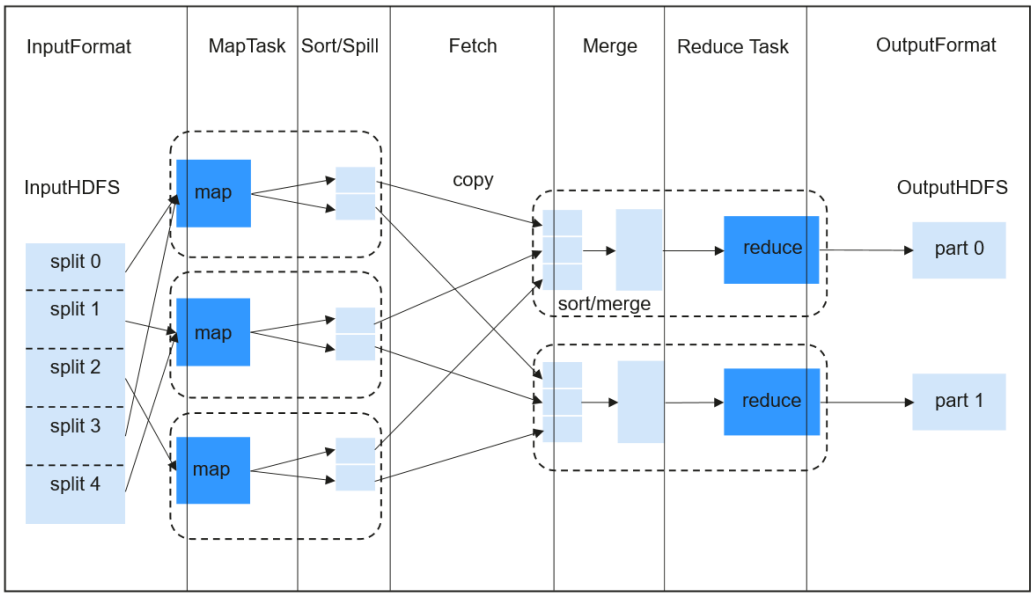
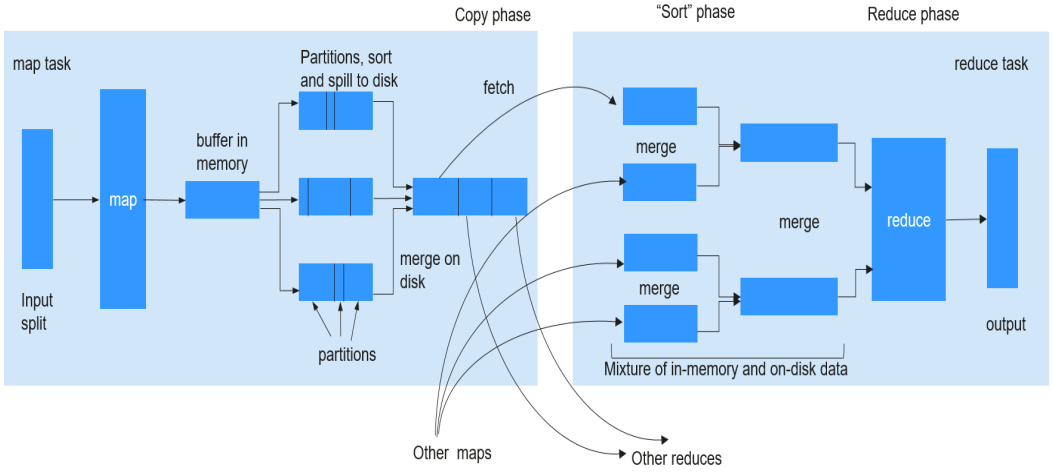


图 6-83 MapReduce 作业执行流程



Reduce过程分为三个不同步骤：Copy、Sort（实际应当称为Merge）及Reduce。在Copy过程中，Reducer尝试从NodeManagers获取Maps的输出并存储在内存或硬盘中。紧接着进行Shuffle过程（包含Sort及Reduce），这个过程将获取到的Maps输出进行存储并有序地合并然后提供给Reducer。当Job有大量的Maps输出需要处理的时候，Shuffle过程将变得非常耗时。对于一些特定的任务（例如hash join或hash aggregation类型的SQL任务），Shuffle过程中的排序并非必须的。但是Shuffle却默认必须进行排序，所以需要对此处进行改进。

此特性通过对MapReduce API进行增强，能自动针对此类型任务关闭Sort过程。当Sort被关闭，获取Maps输出数据以后，直接合并后输出给Reduce，避免了由于排序而浪费大量时间。这种方式极大程度地提升了大部分SQL任务的效率。

MapReduce 开源增强特性：History Server 优化解决日志小文件问题

运行在Yarn上的作业在执行完成后，NodeManager会通过LogAggregationService把产生的日志收集到HDFS上，并从本地文件系统中删除。日志收集到HDFS上以后由

HistoryServer来进行统一的日志管理。LogAggregationService在收集日志时会把container产生的本地日志合并成一个日志文件上传到HDFS，在一定程度上可以减少日志文件的数量。但在规模较大且任务繁忙的集群上，经过长时间的运行，HDFS依然会面临存储的日志文件过多的问题。

以一个20节点的计算场景为例，默认清理周期（15日）内将产生约1800万日志文件，占用NameNode近18G内存空间，同时拖慢HDFS的系统响应速度。

由于收集到HDFS上的日志文件只有读取和删除的需求，因此可以利用Hadoop Archives功能对收集的日志文件目录进行定期归档。

日志归档

在HistoryServer中新增AggregatedLogArchiveService模块，定期检查日志目录中的文件数。在文件数达到设定阈值时，启动归档任务进行日志归档，并在归档完成后删除原日志文件，以减少HDFS上的文件数量。

归档日志清理

由于Hadoop Archives不支持在归档文件中进行删除操作，因此日志清理时需要删除整个归档文件包。通过修改AggregatedLogDeletionService模块，获取归档日志中最新的日志生成时间，若所有日志文件均满足清理条件，则清理该归档日志包。

归档日志浏览

Hadoop Archives支持URI直接访问归档包中的文件内容，因此浏览过程中，当History Server发现原日志文件不存在时，直接将URI重定向到归档文件包中即可访问到已归档的日志文件。

说明

- 本功能通过调用HDFS的Hadoop Archives功能进行日志归档。由于Hadoop Archives归档任务实际上是执行一个MR应用程序，所以在每次执行日志归档任务后，会新增一条MR执行记录。
- 本功能归档的日志来源于日志收集功能，因此只有在日志收集功能开启状态下本功能才会生效。

6.25 MemArtsCC

6.25.1 MemArtsCC 基本原理

MemArtsCC是一款面向存算分离架构的分布式计算侧缓存系统，采用极轻量化的架构设计，部署在计算侧的集群中，通过智能预取远端对象存储上的数据提供高速缓存能力，从而来加速计算任务执行。

MemArtsCC在存储层面将远端对象存储(OBS)上的对象进行切片，并建立索引，大幅提升缓存数据的读取性能。通过ZooKeeper实现轻量化的服务发现，提供超高可用性。基于LRU算法管理分片数据的生命周期。

更多关于MemArtsCC组件操作指导，请参考**使用MemArtsCC**。

MemArtsCC 主要特点

- 去中心化架构，所有实例提供对等服务能力。
- 轻量化设计，极低的资源占用率。

- 应用解耦，业务无需感知无需适配即可使用。
- 高可用，单实例级别异常不影响集群可用性。

MemArtsCC 结构

MemArtsCC由CCSideCar和CCWorker两个角色组成。

在存算架构下，Spark、Hive等计算分析应用的数据存储在对象存储服务(OBS)中。在MemArtsCC集群上一个服务实例称为Worker，对于OBS上的对象数据，Worker缓存其中部分或全部分片到本地的持久化存储（SSD/HDD）中。上层应用通过MemArtsCC SDK读取某个对象时，基于分片索引到特定的Worker上读取分片数据，如果命中缓存则Worker返回对应分片，如果未命中则直接从OBS中读取数据，同时Worker端会异步的加载未命中的分片到本地存储中，供后续使用。

图 6-84 MemArtsCC 结构

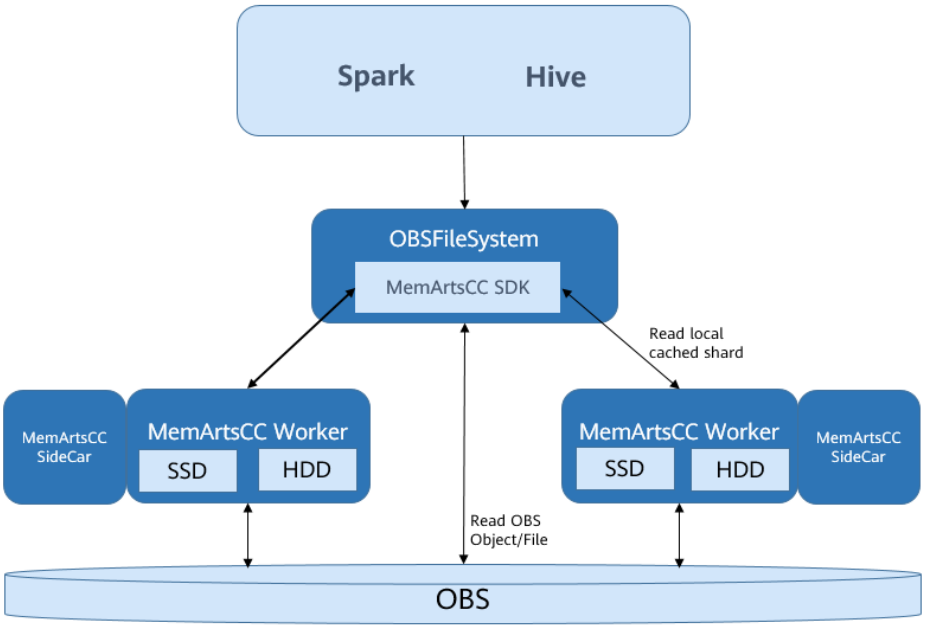


表 6-20 MemArtsCC 结构图说明

名称	说明
MemArtsCC SDK	提供OBSA（OBSA，Hadoop客户端插件）客户端使用的可访问OBS服务器对象的SDK。
CCSideCar	MemArtsCC的管理面服务，提供MemArtsCC服务监控采集、配置下发、服务启停等能力。
CCWorker	MemArtsCC的数据面服务，支持MemArtsCC的缓存数据读写、存储、淘汰等能力。

6.25.2 MemArtsCC 与其他组件的关系

MemArtsCC 与 OBS 的关系

OBS提供一种新的InputStream：OBSMemArtsCCInputStream，该InputStream从部署在计算侧上的MemArtsCC集群读取数据，从而减少OBS服务端压力，提升数据读取性能的目标。

MemArtsCC会将数据持久化存储到计算侧的存储中（SSD），OBS对接MemArtsCC有如下使用场景：

1. 提升存算分离架构访问数据的性能

   利用MemArtsCC的本地存储，访问热点数据不必跨网络，可以提升OBS上层应用数据读取效率。

2. 减少OBS服务端压力

   MemArtsCC会将热点数据存储在计算侧集群，可以起到降低OBS服务端带宽的作用。

MemArtsCC 与 Spark 的关系

Spark从OBS读取数据，OBS会从MemArtsCC读取数据，如果命中则读本地缓存，否则触发预取。

MemArtsCC 与 Hive 的关系

Hive从OBS读取数据，OBS会从MemArtsCC读取数据，如果命中则读本地缓存，否则触发预取。

MemArtsCC 与 HetuEngine 的关系

HetuEngine从OBS读取数据，OBS会从MemArtsCC读取数据，如果命中则读本地缓存，否则触发预取。

6.26 Oozie

Oozie 简介

Oozie是一个基于工作流引擎的开源框架，它能够提供对Hadoop作业的任务调度与协调。

更多关于Oozie组件操作指导，请参考**使用Oozie**。

Oozie 结构

Oozie引擎是一个Web App应用，默认集成到Tomcat中，采用pg数据库。

基于Ext提供WEB Console，该Console仅提供对Oozie工作流的查看和监控功能。通过Oozie对外提REST方式的WS接口，Oozie client通过该接口控制（启动、停止等操作）Workflow流程，从而编排、运行Hadoop MapReduce任务，如**图6-85**所示。

图 6-85 Oozie 框架

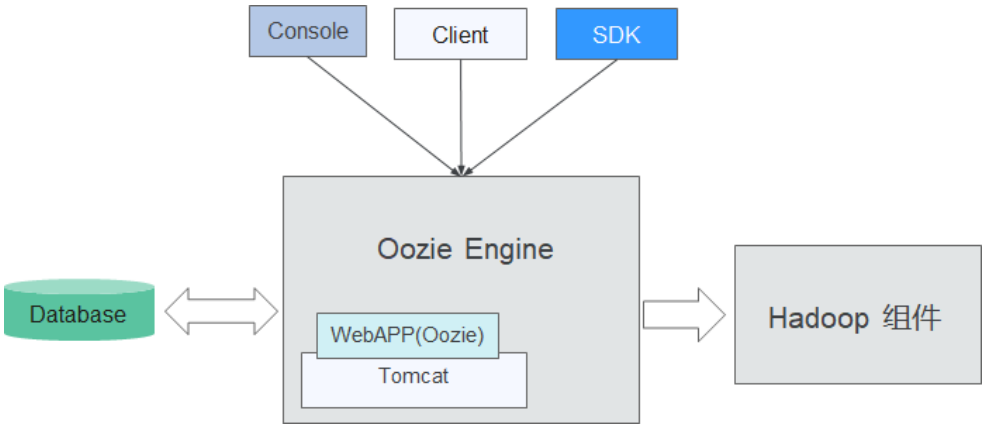


图6-85中各部分的功能说明如表6-21所示。

表 6-21 结构图说明

名称	描述
Console	提供对Oozie流程的查看和监控功能。
Client	通过接口控制Workflow流程：可以执行提交流程，启动流程，运行流程，终止流程，恢复流程等操作。
SDK	软件开发工具包SDK（SoftwareDevelopmentKit）是被软件工程师用于为特定的软件包、软件框架、硬件平台、操作系统等建立应用软件的开发工具的集合。
Database	pg数据库。
WebApp（Oozie）	WebApp（Oozie）即Oozie server，可以用内置的Tomcat容器，也可以用外部的，记录的信息比如日志等放在pg数据库中。
Tomcat	Tomcat服务器是免费的开放源代码的Web应用服务器。
Hadoop组件	底层执行Oozie编排流程的各个组件，包括MapReduce、Hive等。

Oozie 原理

Oozie是一个工作流引擎服务器，用于运行MapReduce任务工作流。同时Oozie还是一个Java Web程序，运行在Tomcat容器中。

Oozie工作流通过HPDL（一种通过XML自定义处理的语言，类似JBOSS JBPM的JPDL）来构造。包含“Control Node”（可控制的工作流节点）、“Action Node”。

- “Control Node”用于控制工作流的编排，如“start”（开始）、“end”（关闭）、“error”（异常场景）、“decision”（选择）、“fork”（并行）、“join”（合并）等。
- Oozie工作流中拥有多个“Action Node”，如MapReduce、Java等。

所有的“Action Node”以有向无环图DAG（Direct Acyclic Graph）的模式部署运行。所以在“Action Node”的运行步骤上是有方向的，当上一个“Action Node”运行完成后才能运行下一个“Action Node”。一旦当前“Action Node”完成，远程服务器将回调Oozie的接口，这时Oozie又会以同样的方式执行工作流中的下一个“Action Node”，直到工作流中所有“Action Node”都完成（完成包括失败）。

Oozie工作流提供各种类型的“Action Node”用于支持不同的业务需要，如MapReduce，HDFS，SSH，Java以及Oozie子流程。

Oozie 开源增强特性

安全增强：

支持Oozie权限管理，提供管理员与普通用户两种角色。

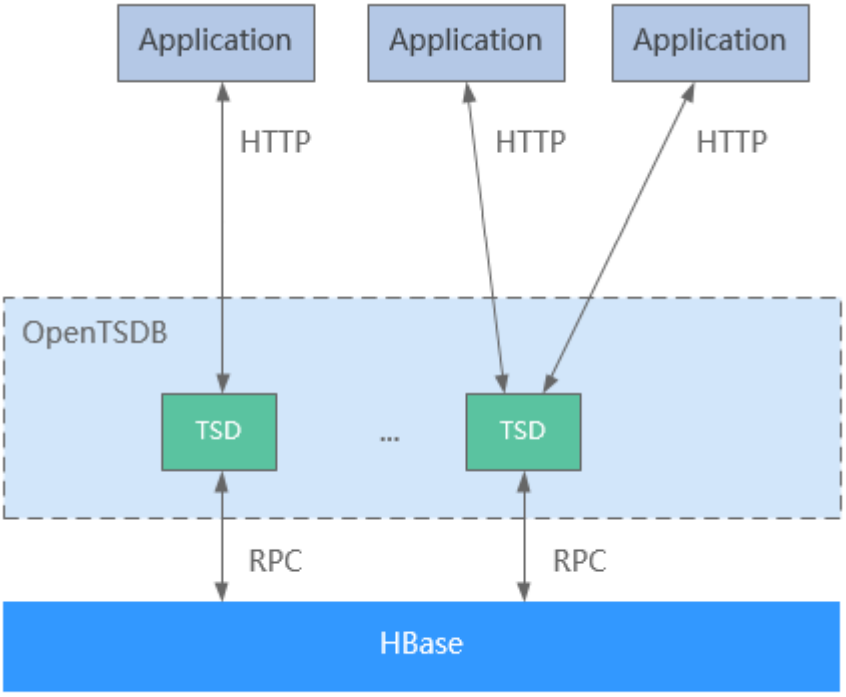
支持单点登录登出，HTTPS访问以及审计日志。

6.27 OpenTSDB

OpenTSDB是一个基于HBase的分布式、可伸缩的时间序列数据库。OpenTSDB的设计目标是用来采集大规模集群中的监控类信息，并可实现数据的秒级查询，解决海量监控类数据在普通数据库中查询存储的局限性。

OpenTSDB由时间序列守护进程（TSD）和一组命令行实用程序组成。与OpenTSDB的交互主要通过运行一个或多个TSD来实现。每个TSD都是独立的。没有主服务器，没有共享状态，因此您可以根据需要运行任意数量的TSD来处理您向其投入的任何负载。每个TSD使用CloudTable集群中的HBase来存储和检索时间序列数据。数据模式经过高度优化，可快速聚合相似的时间序列，从而最大限度地减少存储空间。TSD的用户不需要直接访问底层存储。您可以通过HTTP API与TSD进行通信。所有通信都发生在同一个端口上（TSD通过查看它收到的前几个字节来确定客户端的协议）。

图 6-86 OpenTSDB 架构



OpenTSDB使用场景有如下几个特点：

- 采集指标在某一时间点具有唯一值，没有复杂的结构及关系。
- 监控的指标具有随着时间不断变化的特点。
- 具有HBase的高吞吐，良好的伸缩性等特点。

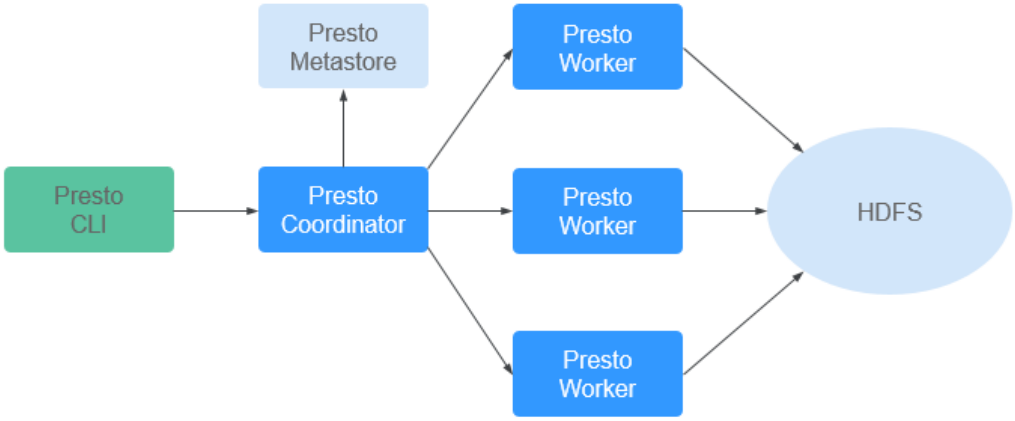
OpenTSDB提供基于HTTP的应用程序编程接口，以实现与外部系统的集成。 几乎所有OpenTSDB功能都可通过API访问，例如查询时间序列数据，管理元数据和存储数据点。详情请参见：**https://opentsdb.net/docs/build/html/api_http/index.html**。

6.28 Presto

Presto是一个开源的用户交互式分析查询的SQL查询引擎，用于针对各种大小的数据源进行交互式分析查询。其主要应用于海量结构化数据/半结构化数据分析、海量多维数据聚合/报表、ETL、Ad-Hoc查询等场景。

Presto允许查询的数据源包括Hadoop分布式文件系统（HDFS），Hive，HBase，Cassandra，关系数据库甚至专有数据存储。一个Presto查询可以组合不同数据源，执行跨数据源的数据分析。

图 6-87 Presto 架构



Presto分布式地运行在一个集群中，包含一个Coordinator和多个Worker进程，查询从客户端（例如CLI）提交到Coordinator，Coordinator进行SQL的解析和生成执行计划，然后分发到多个Worker进程上执行。

有关Presto的详细信息，请参见：**https://prestodb.github.io/**。

Presto 多实例

MRS支持安装Presto多实例，即一个Core/Task节点上安装多个Worker实例，分别为Worker1，Worker2，Worker3…，多个Worker实例共同与Coordinator交互执行计算任务，相比较单实例，能够大大提高节点资源的利用率和计算效率。

Presto多实例仅作用于ARM架构规格，当前单节点最多支持4个实例。

更多Presto部署信息请参考：**https://trino.io/docs/current/installation/deployment.html**。

6.29 Ranger

6.29.1 Ranger 基本原理

Apache Ranger提供一个集中式安全管理框架，提供统一授权和统一审计能力。它可以对整个Hadoop生态中如HDFS、Hive、HBase、Kafka、Storm等进行细粒度的数据访问控制。用户可以利用Ranger提供的前端WebUI控制台通过配置相关策略来控制用户对这些组件的访问权限 。

更多关于Ranger组件操作指导，请参考**使用Ranger**。

Ranger架构如**图6-88**所示

图 6-88 Ranger 结构

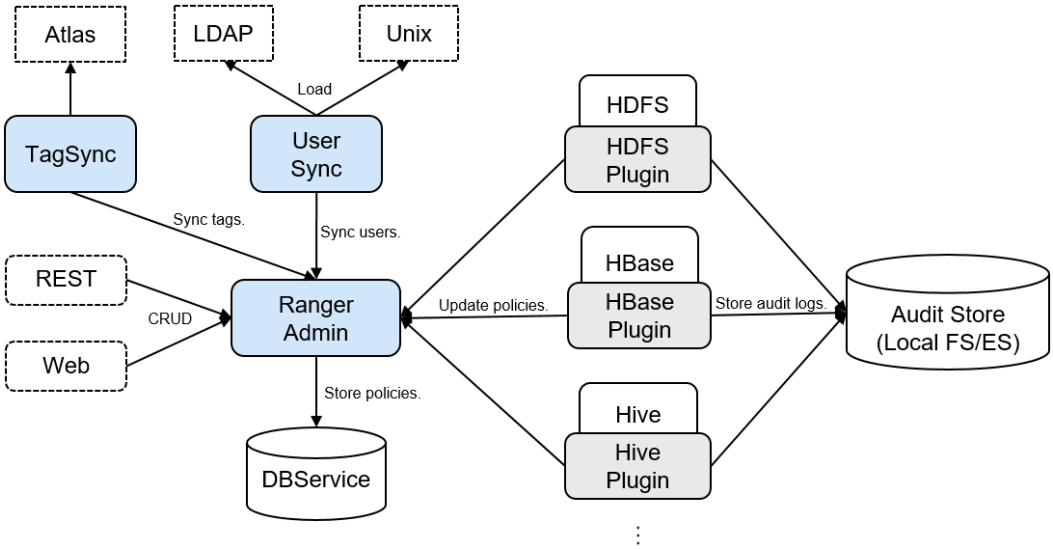


表 6-22 结构图说明

名称	描述
RangerAdmin	Ranger的管理角色，拥有策略管理、用户管理、审计管理等功能，提供WebUI和RestFul接口。
UserSync	负责周期从外部同步用户和用户组信息并写入RangerAdmin中。
TagSync	负责周期从外部Atlas服务同步标签信息并写入RangerAdmin中。

Ranger 原理

- 组件Ranger插件

  Ranger为各组件提供了基于PBAC（Policy-Based Access Control）的权限管理插件，用于替换组件自身原来的鉴权插件。Ranger插件都是由组件侧自身的鉴权接口扩展而来，用户在Ranger WebUI上对指定service设置权限策略，Ranger插件会定期从RangerAdmin处更新策略并缓存在组件本地文件，当有客户端请求需要进行鉴权时，Ranger插件会对请求中携带的用户在策略中进行匹配，随后返回接受或拒绝。

- UserSync用户同步

  UserSync周期性从LDAP/Unix中同步数据到RangerAdmin中，其中安全模式从LDAP中同步，非安全模式从Unix中同步。同步模式默认采取增量模式，每次同步周期UserSync只会更新新增或者变更的用户和用户组，当用户或者用户组被删除时，UserSync不会同步该变更到RangerAdmin，即RangerAdmin中不会同步删除。为了提高性能，UserSync也不会同步没有所属用户的用户组到RangerAdmin中。

- 统一审计

  Ranger插件支持记录审计日志，当前审计日志存储介质支持本地文件。

- 高可靠性

  Ranger支持RangerAdmin双主，两个RangerAdmin同时提供服务，任意一个RangerAdmin故障不会影响Ranger的功能。

- 高性能

  Ranger提供Load-Balance能力，通过浏览器访问Ranger WebUI时Load-Balance会自动选择当前负载较小的RangerAdmin来提供服务。

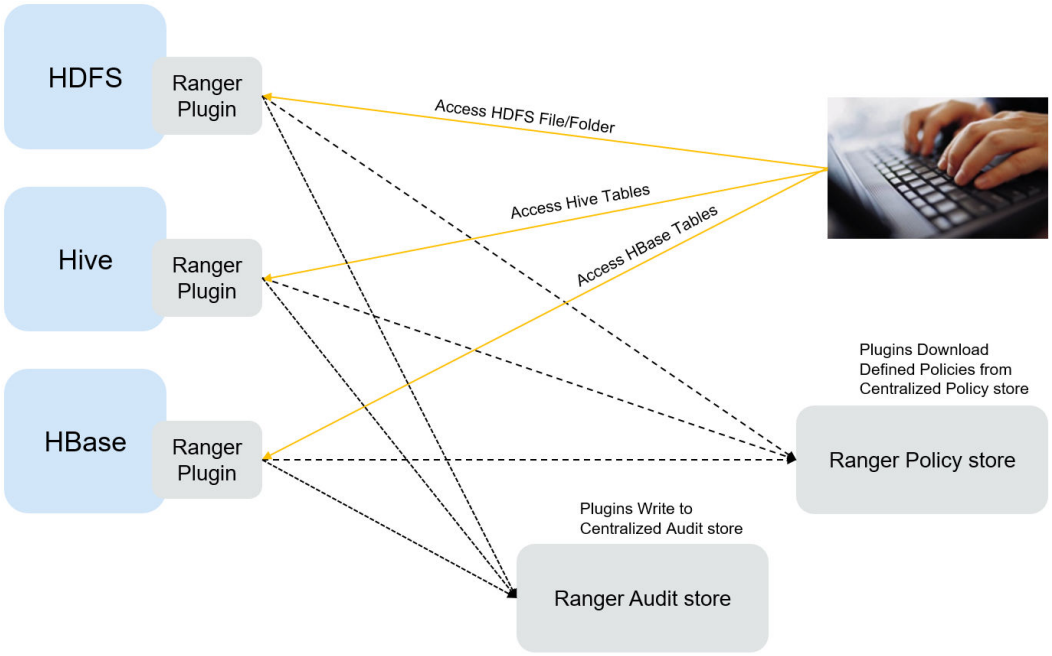
6.29.2 Ranger 与其他组件的关系

Ranger为组件提供基于PBAC的鉴权插件，供组件服务端运行，目前支持Ranger鉴权的组件有HDFS、Yarn、Hive、HBase、Kafka、Storm和Spark等，后续会支持更多组件。

Ranger为各组件提供了基于PBAC（Policy-Based Access Control）的权限管理插件，用于替换组件自身原来的鉴权插件。Ranger插件都是由组件侧自身的鉴权接口扩展而来，用户在Ranger WebUI上对指定service设置权限策略，Ranger插件会定期从RangerAdmin处更新策略并缓存在组件本地文件，当有客户端请求需要进行鉴权时，Ranger插件会对请求中携带的用户在策略中进行匹配，随后返回接受或拒绝。

组件每次启动都会检查组件默认的Ranger Service是否存在，如果不存在则会创建以及为其添加默认Policy。如果用户在使用过程中误删了Service，可以重启或者滚动重启相应组件服务来恢复，如果是误删了默认Policy，可先手动删除Service，再重启组件服务。

图 6-89 Ranger 与组件的关系



6.30 Spark

6.30.1 Spark 基本原理

Spark 简介

Spark是一个开源的，并行数据处理框架，能够帮助用户简单、快速地开发大数据应用，对数据进行离线处理、流式处理、交互式分析等。

Spark提供了一个快速的计算、写入及交互式查询的框架。相比于Hadoop，Spark拥有明显的性能优势。Spark使用in-memory的计算方式，通过这种方式来避免一个MapReduce工作流中的多个任务对同一个数据集进行计算时的IO瓶颈。Spark利用Scala语言实现，Scala能够使得处理分布式数据集时，能够像处理本地化数据一样。除了交互式的数据分析，Spark还能够支持交互式的数据挖掘，由于Spark是基于内存的计算，很方便处理迭代计算，而数据挖掘的问题通常都是对同一份数据进行迭代计算。除此之外，Spark能够运行于安装Hadoop 2.0 Yarn的集群。之所以Spark能够在保留MapReduce容错性，数据本地化，可扩展性等特性的同时，能够保证性能的高效，并且避免繁忙的磁盘IO，主要原因是因为Spark创建了一种叫做RDD（Resilient Distributed Dataset）的内存抽象结构。

原有的分布式内存抽象，例如key-value store以及数据库，支持对于可变状态的细粒度更新，这一点要求集群需要对数据或者日志的更新进行备份来保障容错性。这样就会给数据密集型的工作流带来大量的IO开销。而对于RDD来说，它只有一套受限制的接口，仅支持粗粒度的更新，例如map，join等等。通过这种方式，Spark只需要简单地记录建立数据的转换操作的日志，而不是完整的数据集，就能够提供容错性。这种数据的转换链记录就是数据集的溯源。由于并行程序，通常是对一个大数据集应用相同的计算过程，因此之前提到的粗粒度的更新限制并没有想象中的大。事实上，Spark论文中阐述了RDD完全可以作为多种不同计算框架，例如MapReduce，Pregel等的编程模型。并且，Spark同时提供了操作允许用户显式地将数据转换过程持久化到硬盘。对于数据本地化，是通过允许用户能够基于每条记录的键值，控制数据分区实现的。（采用这种方式的一个明显好处是，能够保证两份需要进行关联的数据将会被同样的方式进行哈希）。如果内存的使用超过了物理限制，Spark将会把这些比较大的分区写入到硬盘，由此来保证可扩展性。

Spark具有如下特点：

- 快速：数据处理能力，比MapReduce快10-100倍。
- 易用：可以通过Java，Scala，Python，简单快速地编写并行的应用处理大数据量，Spark提供了超过80种的操作符来帮助用户组建并行程序。
- 普遍性：Spark提供了众多的工具，例如**Spark SQL**和**Spark Streaming**。可以在一个应用中，方便地将这些工具进行组合。
- 与Hadoop集成：Spark能够直接运行于Hadoop的集群，并且能够直接读取现存的Hadoop数据。

MRS服务的Spark组件具有以下优势：

- MRS服务的Spark Streaming组件支持数据实时处理能力而非定时触发。
- MRS服务的Spark组件支持Structured Streaming，支持DataSet API来构建流式应用，提供了exactly-once的语义支持，流和流的join操作支持内连接和外连接。
- MRS服务的Spark组件支持pandas_udf，可以利用pandas_udf替代pyspark中原来的udf对数据进行处理，可以减少60%-90%的处理时长（受具体操作影响）。
- MRS服务的Spark组件支持Graph功能，支持图计算作业使用图进行建模。
- MRS服务的SparkSQL兼容部分Hive语法（以Hive-Test-benchmark测试集上的64个SQL语句为准）和标准SQL语法（以tpc-ds测试集上的99个SQL语句为准）。

Spark的架构和详细原理介绍，请参见：**https://archive.apache.org/dist/spark/docs/3.1.1/**。

更多关于Spark2x组件操作指导，请参考**使用Spark/Spark2x**。

Spark 结构

Spark的结构如**图6-90**所示，各模块的说明如**表 基本概念说明**所示。

图 6-90 Spark 结构

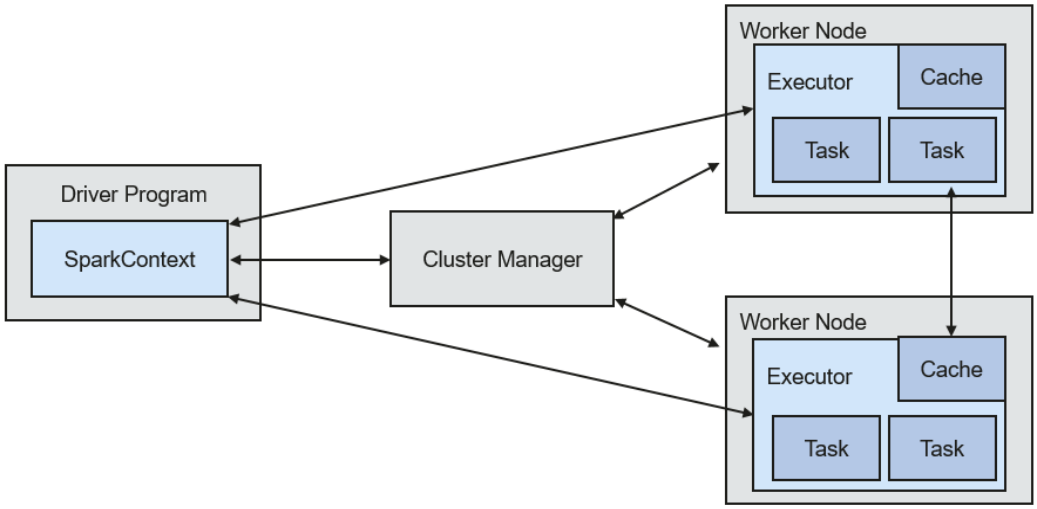


表 6-23 基本概念说明

模块	说明
Cluster Manager	集群管理器，管理集群中的资源。Spark支持多种集群管理器，Spark自带的Standalone集群管理器、Mesos或YARN，系统默认采用YARN模式。
Application	Spark应用，由一个Driver Program和多个Executor组成。
Deploy Mode	部署模式，分为cluster和client模式。cluster模式下，Driver会在集群内的节点运行；而在client模式下，Driver在客户端运行（集群外）。
Driver Program	是Spark应用程序的主进程，运行Application的main()函数并创建SparkContext。负责应用程序的解析、生成Stage并调度Task到Executor上。通常SparkContext代表Driver Program。
Executor	在Work Node上启动的进程，用来执行Task，管理并处理应用中使用到的数据。一个Spark应用一般包含多个Executor，每个Executor接收Driver的命令，并执行一到多个Task。
Worker Node	集群中负责启动并管理Executor以及资源的节点。
Job	一个Action算子（比如collect算子）对应一个Job，由并行计算的多个Task组成。

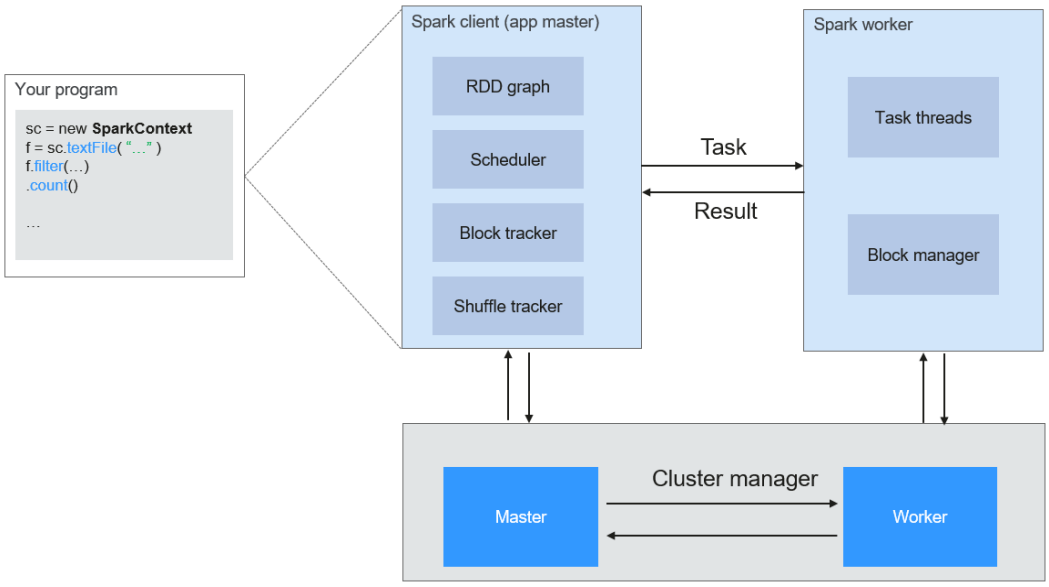
模块	说明
Stage	每个Job由多个Stage组成，每个Stage是一个Task集合，由DAG分割而成。
Task	承载业务逻辑的运算单元，是Spark平台上可执行的最小工作单元。一个应用根据执行计划以及计算量分为多个Task。

Spark 应用运行原理

Spark的应用运行架构如图 Spark应用运行架构所示，运行流程如下所示：

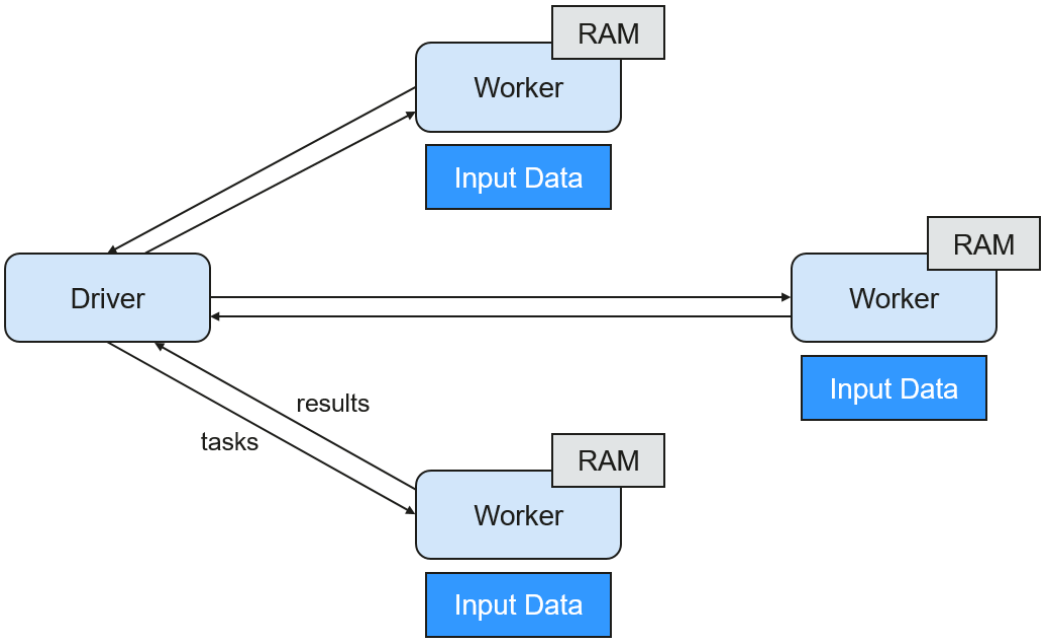
1. 应用程序（Application）是作为一个进程的集合运行在集群上的，由Driver进行协调。
2. 在运行一个应用时，Driver会去连接集群管理器（Standalone、Mesos、YARN）申请运行Executor资源，并启动ExecutorBackend。然后由集群管理器在不同的应用之间调度资源。Driver同时会启动应用程序DAG调度、Stage划分、Task生成。
3. 然后Spark会把应用的代码（传递给SparkContext的JAR或者Python定义的代码）发送到Executor上。
4. 所有的Task执行完成后，用户的应用程序运行结束。

图 6-91 Spark 应用运行架构



Spark采用Master和Worker的模式，如图 Spark的Master和Worker所示。用户在Spark客户端提交应用程序，调度器将Job分解为多个Task发送到各个Worker中执行，各个Worker将计算的结果上报给Driver（即Master），Driver聚合结果返回给客户端。

图 6-92 Spark 的 Master 和 Worker



在此结构中，有几个说明点：

- 应用之间是独立的。

  每个应用有自己的executor进程，Executor启动多个线程，并行地执行任务。无论是在调度方面，或者是executor方面。各个Driver独立调度自己的任务；不同的应用任务运行在不同的JVM上，即不同的Executor。
- 不同Spark应用之间是不共享数据的，除非把数据存储在外部的存储系统上（比如HDFS）。
- 因为Driver程序在集群上调度任务，所以Driver程序建议和worker节点比较近，比如在一个相同的局部网络内。

Spark on YARN有两种部署模式：

- yarn-cluster模式下，Spark的Driver会运行在YARN集群内的ApplicationMaster进程中，ApplicationMaster已经启动之后，提交任务的客户端退出也不会影响任务的运行。
- yarn-client模式下，Driver启动在客户端进程内，ApplicationMaster进程只用来向YARN集群申请资源。

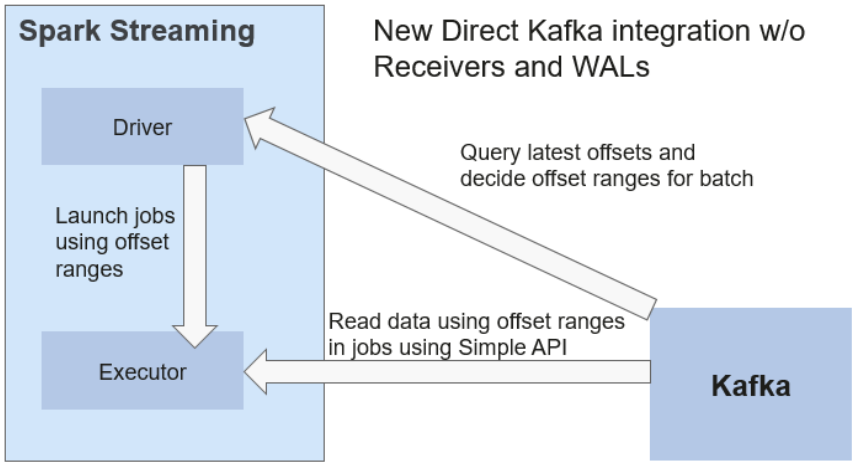
Spark Streaming 原理

Spark Streaming是一种构建在Spark上的实时计算框架，扩展了Spark处理大规模流式数据的能力。当前Spark支持两种数据处理方式：

- Direct Streaming

  Direct Streaming方式主要通过采用Direct API对数据进行处理。以Kafka Direct接口为例，与启动一个Receiver来连续不断地从Kafka中接收数据并写入到WAL中相比，Direct API简单地给出每个batch区间需要读取的偏移量位置。然后，每个batch的Job被运行，而对应偏移量的数据在Kafka中已准备好。这些偏移量信息也被可靠地存储在checkpoint文件中，应用失败重启时可以直接读取偏移量信息。

图 6-93 Direct Kafka 接口数据传输



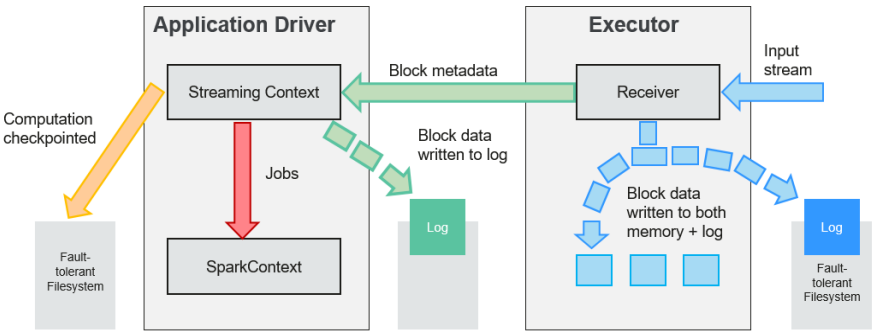
需要注意的是，Spark Streaming可以在失败后重新从Kafka中读取并处理数据段。然而，由于语义仅被处理一次，重新处理的结果和没有失败处理的结果是一致的。

因此，Direct API消除了需要使用WAL和Receivers的情况，且确保每个Kafka记录仅被接收一次，这种接收更加高效。使得Spark Streaming和Kafka可以很好地整合在一起。总体来说，这些特性使得流处理管道拥有高容错性、高效性及易用性，因此推荐使用Direct Streaming方式处理数据。

- Receiver

  在一个Spark Streaming应用开始时（也就是Driver开始时），相关的StreamingContext（所有流功能的基础）使用SparkContext启动Receiver成为长驻运行任务。这些Receiver接收并保存流数据到Spark内存中以供处理。用户传送数据的生命周期如图6-94所示：

  图 6-94 数据传输生命周期

  a. 接收数据（蓝色箭头）

     Receiver将数据流分成一系列小块，存储到Executor内存中。另外，在启用预写日志（Write-ahead Log，简称WAL）以后，数据同时还写入到容错文件系统的预写日志中。

  b. 通知Driver（绿色箭头）

     接收块中的元数据被发送到Driver的StreamingContext。这个元数据包括：

     - 定位其在Executor内存中数据位置的块Reference ID。

- 若启用了WAL，还包括块数据在日志中的偏移信息。

c. 处理数据（红色箭头）

对每个批次的数据，StreamingContext使用Block信息产生RDD及其Job。StreamingContext通过运行任务处理Executor内存中的Block来执行Job。

d. 周期性的设置检查点（橙色箭头）

为了容错的需要，StreamingContext会周期性的设置检查点，并保存到外部文件系统中。

容错性

Spark及其RDD允许无缝地处理集群中任何Worker节点的故障。鉴于Spark Streaming建立于Spark之上，因此其Worker节点也具备了同样的容错能力。然而，由于Spark Streaming的长期正常运行需求，其应用程序必须也具备从Driver进程（协调各个Worker的主要应用进程）故障中恢复的能力。使Spark Driver能够容错是件很棘手的事情，因为可能是任意计算模式实现的任意用户程序。不过Spark Streaming应用程序在计算上有一个内在的结构：在每批次数据周期性地执行同样的Spark计算。这种结构允许把应用的状态（也叫做Checkpoint）周期性地保存到可靠的存储空间中，并在Driver重新启动时恢复该状态。

对于文件这样的源数据，这个Driver恢复机制足以做到零数据丢失，因为所有的数据都保存在了像HDFS这样的容错文件系统中。但对于像Kafka和Flume等其他数据源，有些接收到的数据还只缓存在内存中，尚未被处理，就有可能会丢失。这是由于Spark应用的分布操作方式引起的。当Driver进程失败时，所有在Cluster Manager中运行的Executor，连同在内存中的所有数据，也同时被终止。为了避免这种数据损失，Spark Streaming引进了WAL功能。

WAL通常被用于数据库和文件系统中，用来保证任何数据操作的持久性，即先将操作记入一个持久的日志，再对数据施加这个操作。若施加操作的过程中执行失败了，则通过读取日志并重新施加前面指定的操作，系统就得到了恢复。下面介绍了如何利用这样的概念保证接收到的数据的持久性。

Kafka数据源使用Receiver来接收数据，是Executor中的长运行任务，负责从数据源接收数据，并且在数据源支持时还负责确认收到数据的结果（收到的数据被保存在Executor的内存中，然后Driver在Executor中运行来处理任务）。

当启用了预写日志以后，所有收到的数据同时还保存到了容错文件系统的日志文件中。此时即使Spark Streaming失败，这些接收到的数据也不会丢失。另外，接收数据的正确性只在数据被预写到日志以后Receiver才会确认，已经缓存但还没有保存的数据可以在Driver重新启动之后由数据源再发送一次。这两个机制确保了零数据丢失，即所有的数据或者从日志中恢复，或者由数据源重发。

如果需要启用预写日志功能，可以通过如下动作实现：

- 通过“streamingContext.checkpoint”设置checkpoint的目录，这个目录是一个HDFS的文件路径，既用作保存流的checkpoint，又用作保存预写日志。
- 设置SparkConf的属性“spark.streaming.receiver.writeAheadLog.enable”为“true”（默认值是“false”）。

在WAL被启用以后，所有Receiver都获得了能够从可靠收到的数据中恢复的优势。建议缓存RDD时不采取多备份选项，因为用于预写日志的容错文件系统很可能也复制了数据。

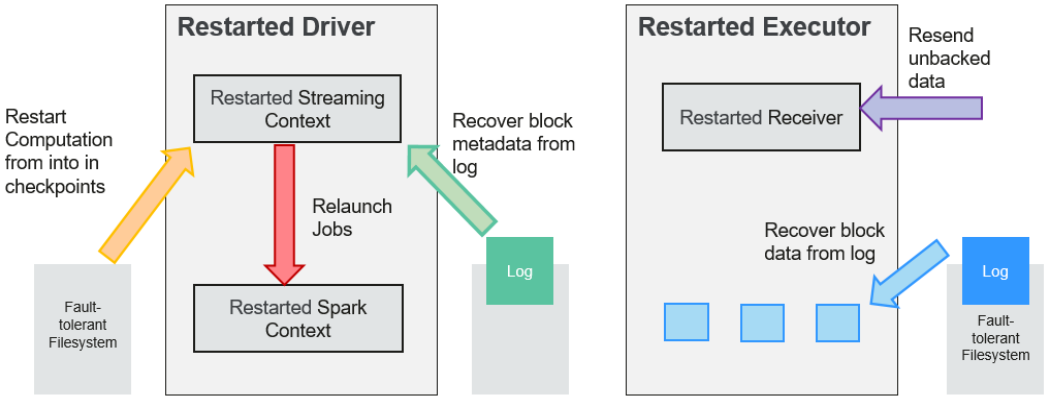
说明

在启用了预写日志以后，数据接收吞吐率会有降低。由于所有数据都被写入容错文件系统，文件系统的写入吞吐率和用于数据复制的网络带宽，可能就是潜在的瓶颈了。在此情况下，建议创建更多的Receiver增加数据接收的并行度，或使用更好的硬件以增加容错文件系统的吞吐率。

恢复流程

当一个失败的Driver重启时，按如下流程启动：

图 6-95 计算恢复流程



1. 恢复计算（橙色箭头）

   使用checkpoint信息重启Driver，重新构造SparkContext并重启Receiver。

2. 恢复元数据块（绿色箭头）

   为了保证能够继续下去所必备的全部元数据块都被恢复。

3. 未完成作业的重新形成（红色箭头）

   由于失败而没有处理完成的批处理，将使用恢复的元数据再次产生RDD和对应的作业。

4. 读取保存在日志中的块数据（蓝色箭头）

   在这些作业执行时，块数据直接从预写日志中读出。这将恢复在日志中可靠地保存的所有必要数据。

5. 重发尚未确认的数据（紫色箭头）

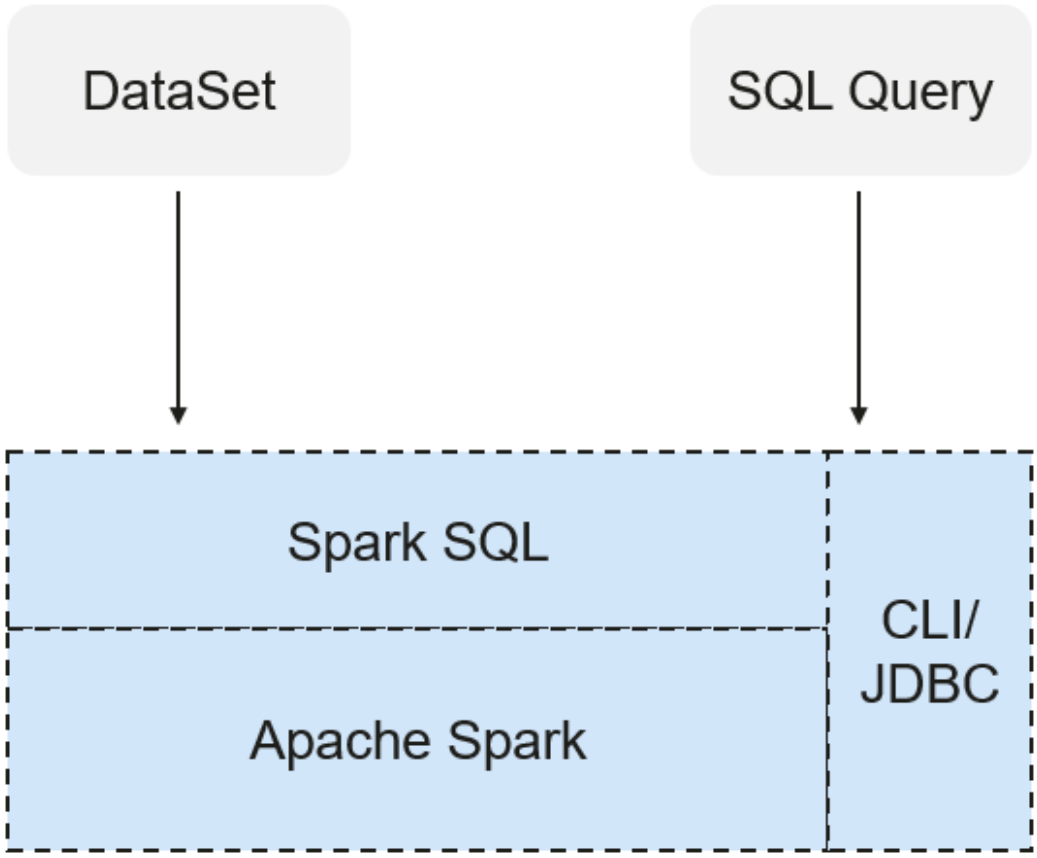
   失败时没有保存到日志中的缓存数据将由数据源再次发送。因为Receiver尚未对其确认。

因此通过预写日志和可靠的Receiver，Spark Streaming就可以保证没有输入数据会由于Driver的失败而丢失。

SparkSQL 和 DataSet 原理

SparkSQL

图 6-96 SparkSQL 和 DataSet



Spark SQL是Spark中用于结构化数据处理的模块。在Spark应用中，可以无缝地使用SQL语句亦或是DataSet API对结构化数据进行查询。

Spark SQL以及DataSet还提供了一种通用的访问多数据源的方式，可访问的数据源包括Hive、CSV、Parquet、ORC、JSON和JDBC数据源，这些不同的数据源之间也可以实现互相操作。Spark SQL复用了Hive的前端处理逻辑和元数据处理模块，使用Spark SQL可以直接对已有的Hive数据进行查询。

另外，SparkSQL还提供了诸如API、CLI、JDBC等诸多接口，对客户端提供多样接入形式。

Spark SQL Native DDL/DML

Spark 1.5版本将很多DDL/DML命令下压到Hive执行，造成了与Hive的耦合，且在一定程度上不够灵活（比如报错不符合预期、结果与预期不一致等）。

Spark 3.1.1版本实现了命令的本地化，使用Spark SQL Native DDL/DML取代Hive执行DDL/DML命令。一方面实现和Hive的解耦，另一方面可以对命令进行定制化。

DataSet

DataSet是一个由特定域的对象组成的强类型集合，可通过功能或关系操作并行转换其中的对象。 每个Dataset还有一个非类型视图，即由多个列组成的DataSet，称为DataFrame。

DataFrame是一个由多个列组成的结构化的分布式数据集合，等同于关系数据库中的一张表，或者是R/Python中的data frame。DataFrame是Spark SQL中的最基本的概

念，可以通过多种方式创建，例如结构化的数据集、Hive表、外部数据库或者是RDD。

可用于DataSet的操作分为Transformation和Action：

- Transformation操作可生成新的DataSet。
  如map、filter、select和aggregate (groupBy)。
- Action操作可触发计算及返回记结果。
  如count、show或向文件系统写数据。

通常使用以下两种方法创建一个DataSet：

- 最常见的方法是通过使用SparkSession上的read函数将Spark指向存储系统上的某些文件。
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people = spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- 还可通过已存在的DataSet上可用的transformation操作来创建数据集。
  例如，在已存在的DataSet上应用map操作来创建新的DataSet：
  ```
  val names = people.map(_.name)  // 使用Scala语言，且names为一个Dataset
  Dataset<String> names = people.map((Person p) -> p.name, Encoders.STRING)); // Java
  ```

CLI和JDBCServer

除了API编程接口之外，Spark SQL还对外提供CLI/JDBC接口：

- spark-shell和spark-sql脚本均可以提供CLI，以便于调试。
- JDBCServer提供JDBC接口，外部可直接通过发送JDBC请求来完成结构化数据的计算和解析。

SparkSession 原理

SparkSession是Spark编程的统一API，也可看作是读取数据的统一入口。SparkSession提供了一个统一的入口点来执行以前分散在多个类中的许多操作，并且还为那些较旧的类提供了访问器方法，以实现最大的兼容性。

使用构建器模式创建SparkSession。如果存在SparkSession，构建器将自动重用现有的SparkSession；如果不存在则会创建一个SparkSession。 在I/O期间，在构建器中设置的配置项将自动同步到Spark和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可以用于对数据执行SQL查询，将结果返回为DataFrame。
  ```
  sparkSession.sql("select * from person").show
  ```
- SparkSession可以用于设置运行时的配置项，这些配置项可以在SQL中使用变量替换。
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- SparkSession包括一个“catalog”方法，其中包含使用Metastore（即数据目录）的方法。方法返回值为数据集，可以使用相同的Dataset API来运行。
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```

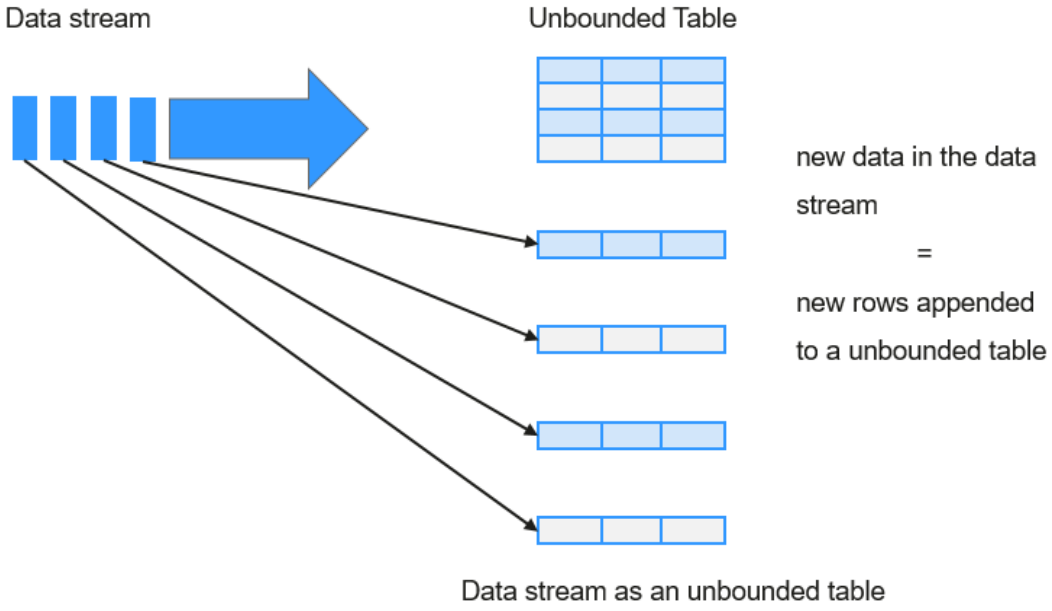

- 底层SparkContext可以通过SparkSession的SparkContext API访问。
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Structured Streaming 原理

Structured Streaming是构建在Spark SQL引擎上的流式数据处理引擎，用户可以使用Scala、Java、Python或R中的Dataset/DataFrame API进行流数据聚合运算、按事件时间窗口计算、流流Join等操作。当流数据连续不断地产生时，Spark SQL将会增量的、持续不断地处理这些数据并将结果更新到结果集中。同时，系统通过checkpoint和Write Ahead Logs确保端到端的完全一次性容错保证。

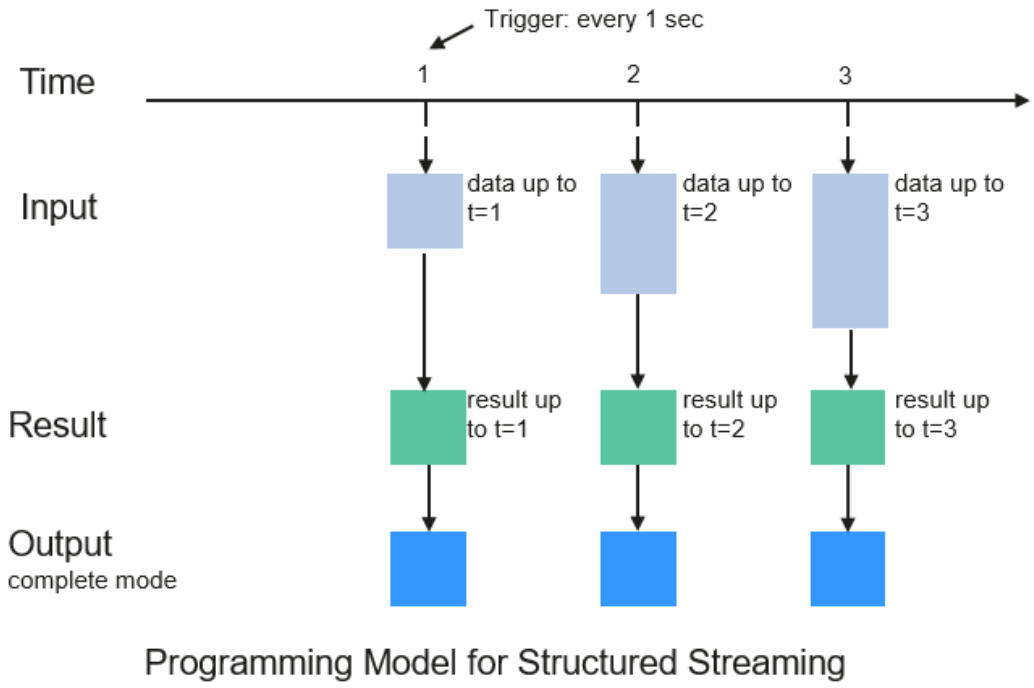
Structured Streaming的核心是将流式的数据看成一张不断增加的数据库表，这种流式的数据处理模型类似于数据块处理模型，可以把静态数据库表的一些查询操作应用在流式计算中，Spark执行标准的SQL查询，从不断增加的无边界表中获取数据。

图 6-97 Structured Streaming 无边界表



每一条查询的操作都会产生一个结果集Result Table。每一个触发间隔，当新的数据新增到表中，都会最终更新Result Table。无论何时结果集发生了更新，都能将变化的结果写入一个外部的存储系统。

图 6-98 Structured Streaming 数据处理模型



Structured Streaming在OutPut阶段可以定义不同的存储方式，有如下3种：

- Complete Mode：整个更新的结果集都会写入外部存储。整张表的写入操作将由外部存储系统的连接器完成。
- Append Mode：当时间间隔触发时，只有在Result Table中新增加的数据行会被写入外部存储。这种方式只适用于结果集中已经存在的内容不希望发生改变的情况下，如果已经存在的数据会被更新，不适合使用此种方式。
- Update Mode：当时间间隔触发时，只有在Result Table中被更新的数据才会被写入外部存储系统。注意，和Complete Mode方式的不同之处是不更新的结果集不会写入外部存储。

Spark 常见基本概念

- **RDD**

  即弹性分布数据集（Resilient Distributed Dataset），是Spark的核心概念。指的是一个只读的，可分区的分布式数据集，这个数据集的全部或部分可以缓存在内存中，在多次计算间重用。

  RDD的生成：

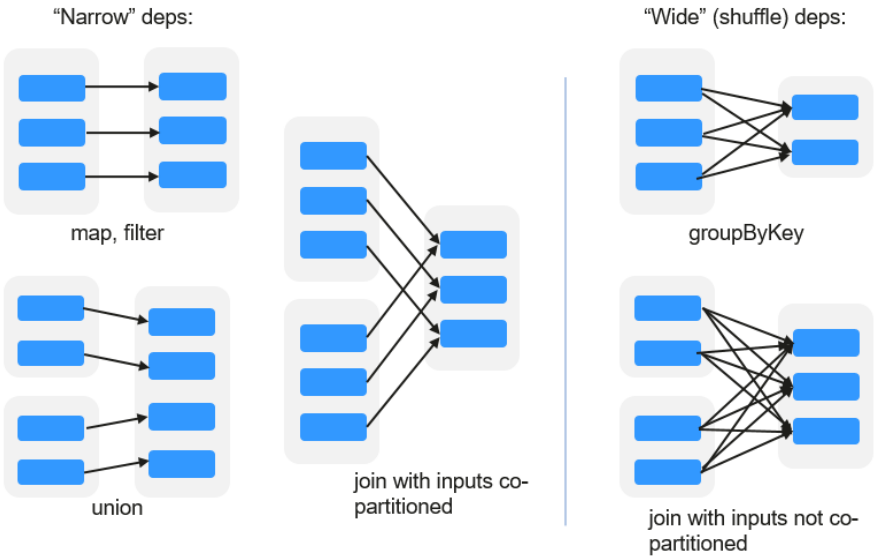
  - 从HDFS输入创建，或从与Hadoop兼容的其他存储系统中输入创建。
  - 从父RDD转换得到新RDD。
  - 从数据集合转换而来，通过编码实现。

  RDD的存储：

  - 用户可以选择不同的存储级别缓存RDD以便重用（RDD有11种存储级别）。
  - 当前RDD默认是存储于内存，但当内存不足时，RDD会溢出到磁盘中。
- **Dependency（RDD的依赖）**

  RDD的依赖分别为：窄依赖和宽依赖。

图 6-99 RDD 的依赖



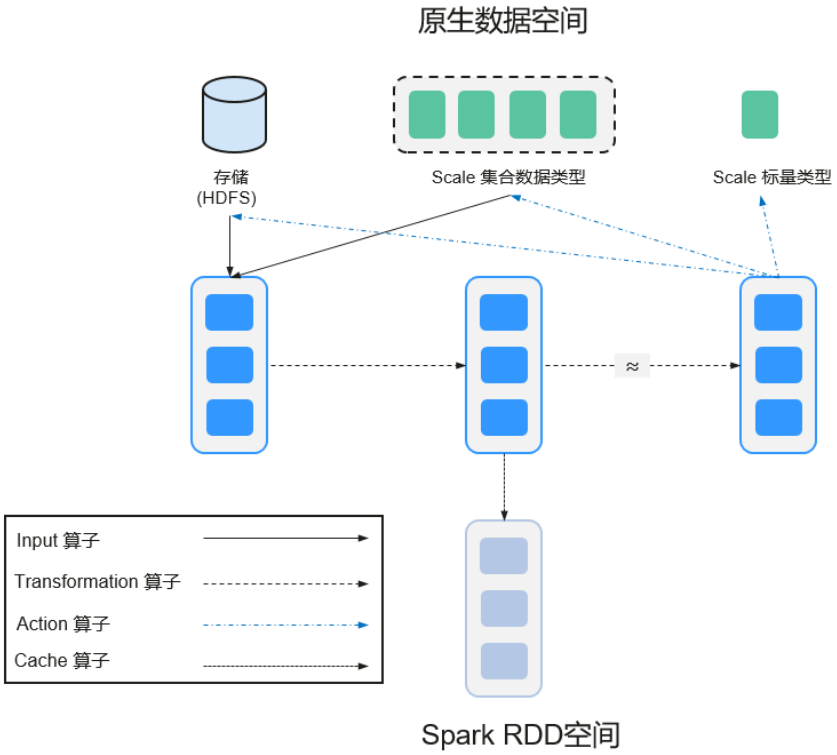
- **窄依赖：**指父RDD的每一个分区最多被一个子RDD的分区所用。
- **宽依赖：**指子RDD的分区依赖于父RDD的所有分区。

窄依赖对优化很有利。逻辑上，每个RDD的算子都是一个fork/join（此join非上文的join算子，而是指同步多个并行任务的barrier）：把计算fork到每个分区，算完后join，然后fork/join下一个RDD的算子。如果直接翻译到物理实现，是很不经济的：一是每一个RDD（即使是中间结果）都需要物化到内存或存储中，费时费空间；二是join作为全局的barrier，是很昂贵的，会被最慢的那个节点拖死。如果子RDD的分区到父RDD的分区是窄依赖，就可以实施经典的fusion优化，把两个fork/join合为一个；如果连续的变换算子序列都是窄依赖，就可以把很多个fork/join并为一个，不但减少了大量的全局barrier，而且无需物化很多中间结果RDD，这将极大地提升性能。Spark把这个叫做流水线（pipeline）优化。

- **Transformation和Action（RDD的操作）**

  对RDD的操作包含Transformation（返回值还是一个RDD）和Action（返回值不是一个RDD）两种。RDD的操作流程如图6-100所示。其中Transformation操作是Lazy的，也就是说从一个RDD转换生成另一个RDD的操作不是马上执行，Spark在遇到Transformations操作时只会记录需要这样的操作，并不会去执行，需要等到有Actions操作的时候才会真正启动计算过程进行计算。Actions操作会返回结果或把RDD数据写到存储系统中。Actions是触发Spark启动计算的动因。

图 6-100 RDD 操作示例



RDD看起来与Scala集合类型没有太大差别，但数据和运行模型大相迥异。

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

a. textFile算子从HDFS读取日志文件，返回file（作为RDD）。

b. filter算子筛出带“ERROR”的行，赋给errors（新RDD）。filter算子是一个Transformation操作。

c. cache算子缓存下来以备未来使用。

d. count算子返回errors的行数。count算子是一个Action操作。

Transformation操作可以分为如下几种类型：

- 视RDD的元素为简单元素。

  输入输出一对一，且结果RDD的分区结构不变，主要是map。

  输入输出一对多，且结果RDD的分区结构不变，如flatMap（map后由一个元素变为一个包含多个元素的序列，然后展平为一个个的元素）。

  输入输出一对一，但结果RDD的分区结构发生了变化，如union（两个RDD合为一个，分区数变为两个RDD分区数之和）、coalesce（分区减少）。

  从输入中选择部分元素的算子，如filter、distinct（去除重复元素）、subtract（本RDD有、其他RDD无的元素留下来）和sample（采样）。

- 视RDD的元素为Key-Value对。

  对单个RDD做一对一运算，如mapValues（保持源RDD的分区方式，这与map不同）；

  对单个RDD重排，如sort、partitionBy（实现一致性的分区划分，这个对数据本地性优化很重要）；

对单个RDD基于key进行重组和reduce，如groupByKey、reduceByKey；

对两个RDD基于key进行join和重组，如join、cogroup。

说明

后三种操作都涉及重排，称为shuffle类操作。

Action操作可以分为如下几种：

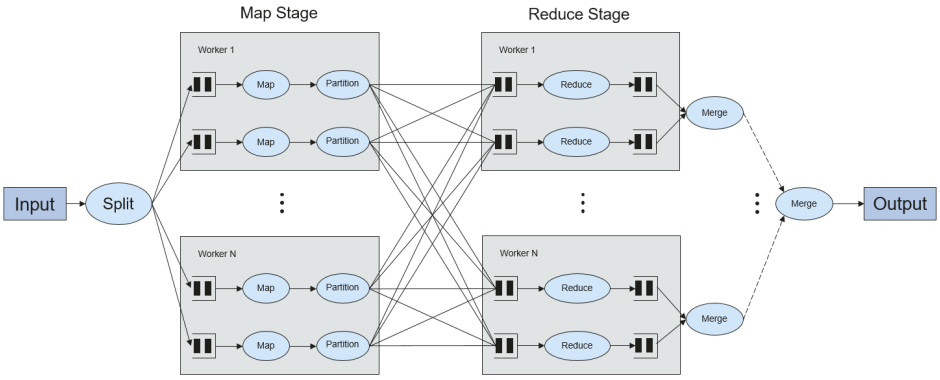
- 生成标量，如count（返回RDD中元素的个数）、reduce、fold/aggregate（返回几个标量）、take（返回前几个元素）。
- 生成Scala集合类型，如collect（把RDD中的所有元素导入Scala集合类型）、lookup（查找对应key的所有值）。
- 写入存储，如与前文textFile对应的saveAsTextFile。
- 还有一个检查点算子checkpoint。当Lineage特别长时（这在图计算中时常发生），出错时重新执行整个序列要很长时间，可以主动调用checkpoint把当前数据写入稳定存储，作为检查点。

- **Shuffle**

  Shuffle是MapReduce框架中的一个特定的phase，介于Map phase和Reduce phase之间，当Map的输出结果要被Reduce使用时，每一条输出结果需要按key哈希，并且分发到对应的Reducer上去，这个过程就是shuffle。由于shuffle涉及到了磁盘的读写和网络的传输，因此shuffle性能的高低直接影响到了整个程序的运行效率。

  下图清晰地描述了MapReduce算法的整个流程。

  图 6-101 算法流程



  概念上shuffle就是一个沟通数据连接的桥梁，实际上shuffle这一部分是如何实现的呢，下面就以Spark为例讲解shuffle在Spark中的实现。

  Shuffle操作将一个Spark的Job分成多个Stage，前面的stages会包括一个或多个ShuffleMapTasks，最后一个stage会包括一个或多个ResultTask。

- **Spark Application的结构**

  Spark Application的结构可分为两部分：初始化SparkContext和主体程序。

  - 初始化SparkContext：构建Spark Application的运行环境。

    构建SparkContext对象，如：

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

    参数介绍：

    master：连接字符串，连接方式有local、yarn-cluster、yarn-client等。

appName：构建的Application名称。

SparkHome：集群中安装Spark的目录。

jars：应用程序代码和依赖包。

- 主体程序：处理数据

提交Application的描述请参见：**https://archive.apache.org/dist/spark/docs/3.1.1/submitting-applications.html**

- **Spark shell命令**

  Spark基本shell命令，支持提交Spark应用。命令为：

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

  参数解释：

  --class：Spark应用的类名。

  --master：Spark用于所连接的master，如yarn-client，yarn-cluster等。

  application-jar：Spark应用的jar包的路径。

  application-arguments：提交Spark应用的所需要的参数（可以为空）。

- **Spark JobHistory Server**

  用于监控正在运行的或者历史的Spark作业在Spark框架各个阶段的细节以及提供日志显示，帮助用户更细粒度地去开发、配置和调优作业。

6.30.2 Spark HA 方案介绍

Spark 多主实例 HA 原理与实现方案

基于社区已有的JDBCServer基础上，采用多主实例模式实现了其高可用性方案。集群中支持同时共存多个JDBCServer服务，通过客户端可以随机连接其中的任意一个服务进行业务操作。即使集群中一个或多个JDBCServer服务停止工作，也不影响用户通过同一个客户端接口连接其他正常的JDBCServer服务。

多主实例模式相比主备模式的HA方案，优势主要体现在对以下两种场景的改进。

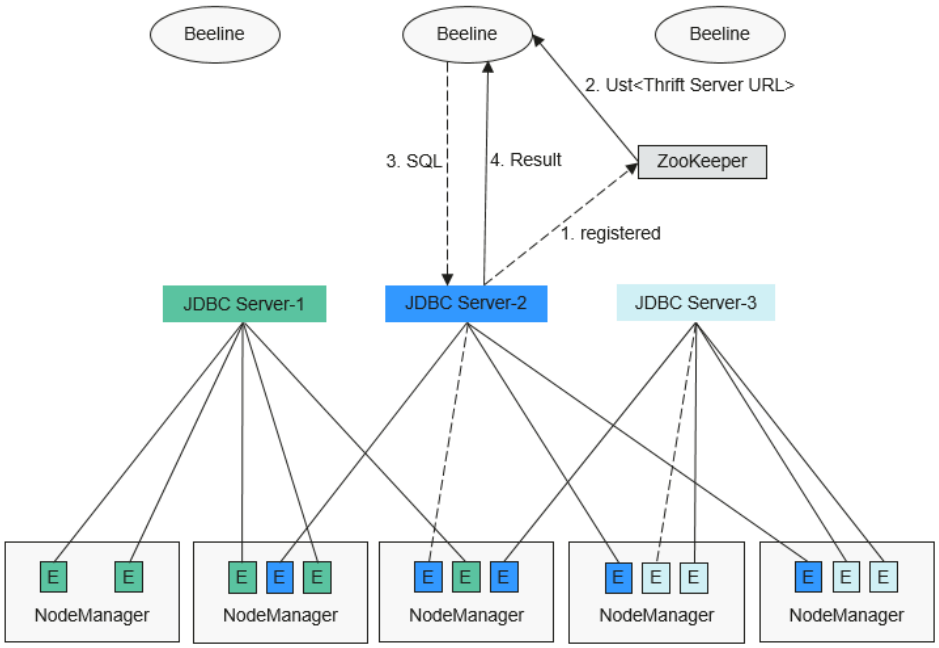
- 主备模式下，当发生主备切换时，会存在一段时间内服务不可用，该时间JDBCServer无法控制，取决于Yarn服务的资源情况。
- Spark中通过类似于HiveServer2的Thrift JDBC提供服务，用户通过Beeline以及JDBC接口访问。因此JDBCServer集群的处理能力取决于主Server的单点能力，可扩展性不够。

采用多主实例模式的HA方案，不仅可以规避主备切换服务中断的问题，实现服务不中断或少中断，还可以通过横向扩展集群来提高并发能力。

- **实现方案**

  多主实例模式的HA方案原理如下图所示。

图 6-102 Spark JDBCServer HA



1. JDBCServer在启动时，向ZooKeeper注册自身消息，在指定目录中写入节点，节点包含了该实例对应的IP，端口，版本号和序列号等信息。
2. 客户端连接JDBCServer时，需要指定Namespace，即访问ZooKeeper哪个目录下的JDBCServer实例。在连接的时候，会从Namespace下随机选择一个实例连接。
3. 客户端成功连接JDBCServer服务后，向JDBCServer服务发送SQL语句。
4. JDBCServer服务执行客户端发送的SQL语句后，将结果返回给客户端。

在HA方案中，每个JDBCServer实例都是独立且等同的，当其中一个实例在升级或者业务中断时，其他的实例也能接受客户端的连接请求。

多主实例方案遵循以下规则：

- 当一个实例异常退出时，其他实例不会接管此实例上的会话，也不会接管此实例上运行的业务。
- 当JDBCServer进程停止时，删除在ZooKeeper上的相应节点。
- 由于客户端选择服务端的策略是随机的，可能会出现会话随机分配不均匀的情况，进而可能引起实例间的负载不均衡。
- 实例进入维护模式（即进入此模式后不再接受新的客户端连接）后，当达到退服超时时间，仍在此实例上运行的业务有可能会发生失败。
- **URL连接介绍**
  - 多主实例模式

    多主实例模式的客户端读取ZooKeeper节点中的内容，连接对应的JDBCServer服务。连接字符串为：

    - 安全模式下：

      Kinit认证方式下的JDBCURL如下所示：

      ```
      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.<系统域名>@<系统域名>;
      ```

说明

- 其中“<zkNode_IP>:<zkNode_Port>”是ZooKeeper的URL，多个URL以逗号隔开。

  例如：
  “192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181”。
- 其中“sparkthriftserver2x”是ZooKeeper上的目录，表示客户端从该目录下随机选择JDBCServer实例进行连接。

示例：安全模式下通过Beeline客户端连接时执行以下命令：

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.*<系统域名>*@*<系统域名>*;"**

Keytab认证方式下的JDBCURL如下所示：

```
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.<系统域名>@<系统域名>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
```

其中<principal_name>表示用户使用的Kerberos用户的principal，如“test@*<系统域名>*”。<path_to_keytab>表示<principal_name>对应的keytab文件路径，如“/opt/auth/test/user.keytab”。

- 普通模式下：

  ```
  jdbc:hive2://<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
  ```

  示例：普通模式下通过Beeline客户端连接时执行以下命令：

  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;"**

- 非多主实例模式

  非多主实例模式的客户端连接的是某个指定JDBCServer节点。该模式的连接字符串相比多主实例模式的去掉关于Zookeeper的参数项“serviceDiscoveryMode”和“zooKeeperNamespace”。

  示例：安全模式下通过Beeline客户端连接非多主实例模式时执行以下命令：

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://** *<server_IP>:<server_Port>*/**;user.principal=spark/hadoop.*<系统域名>*@*<系统域名>*;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.*<系统域名>*@*<系统域名>*;"**

  说明

  - 其中“<server_IP>:<server_Port>”是指定JDBCServer节点的URL。
  - “CLIENT_HOME”是指客户端路径。

  多主实例模式与非多主实例模式两种模式的JDBCServer接口相比，除连接方式不同外其他使用方法相同。由于Spark JDBCServer是Hive中的HiveServer2的另外一个实现，其使用方法，请参见**https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**。

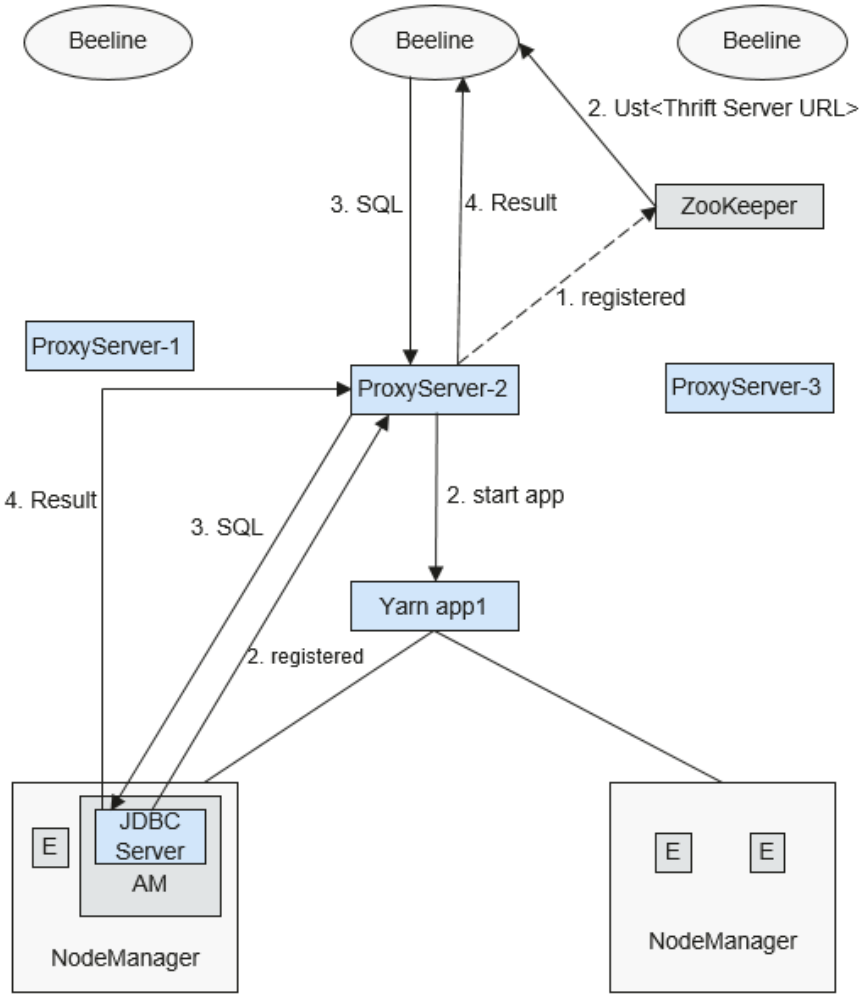
Spark 多租户 HA 方案实现

JDBCServer多主实例方案中，JDBCServer实现使用YARN-Client模式，但YARN资源队列只有一个，为了解决这种资源局限的问题，引入了多租户模式。

多租户模式是将JDBCServer和租户绑定，每一个租户对应一个或多个JDBCServer，而一个JDBCServer只给一个租户提供服务。不同的租户可以配置不同的YARN队列，从而达到资源隔离，且JDBCServer根据需求动态启动，可避免浪费资源。

- **实现方案**
  多租户模式的HA方案原理如图6-103所示。

  图 6-103 Spark JDBCServer 多租户



a. ProxyServer在启动时，向ZooKeeper注册自身消息，在指定目录中写入节点信息，节点信息包含了该实例对应的IP，端口，版本号和序列号等信息。

   说明

   多租户模式下，JDBCServer实例是指ProxyServer（JDBCServer代理）。

b. 客户端连接ProxyServer时，需要指定Namespace，即访问ZooKeeper哪个目录下的ProxyServer实例。在连接的时候，会从Namespace下随机选择一个实例连接，详细URL参见URL连接介绍。

c. 客户端成功连接ProxyServer服务，ProxyServer服务首先确认是否有该租户的JDBCServer存在，如果有，直接将Beeline连上真正的JDBCServer；如果没有，则以YARN-Cluster模式启动一个新的JDBCServer。JDBCServer启动成功后，ProxyServer会获取JDBCServer的地址，并将Beeline连上JDBCServer。

d. 客户端发送SQL语句给ProxyServer，ProxyServer将语句转交给真正连上的JDBCServer处理。最后JDBCServer服务将结果返回给ProxyServer，ProxyServer再将结果返回给客户端。

在HA方案中，每个ProxyServer服务（即实例）都是独立且等同的，当其中一个实例在升级或者业务中断时，其他的实例也能接受客户端的连接请求。

- **URL连接介绍**
  - 多租户模式

    多租户模式的客户端读取ZooKeeper节点中的内容，连接对应的ProxyServer服务。连接字符串为：

    - 安全模式下：

      Kinit认证方式下的客户端URL如下所示：

      ```
      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.<系统域名>@<系统域名>;
      ```

      说明

      - 其中“<zkNode_IP>:<zkNode_Port>”是ZooKeeper的URL，多个URL以逗号隔开。

        例如：
        “192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181”。
      - 其中sparkthriftserver2x是ZooKeeper上的目录，表示客户端从该目录下随机选择JDBCServer实例进行连接。

      示例：安全模式下通过Beeline客户端连接时执行以下命令：

      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.<系统域名>@<系统域名>;"**

      Keytab认证方式下的URL如下所示：

      ```
      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.<系统域名>@<系统域名>;user.principal=<principal_name>;user.keytab=<path_to_keytab>
      ```

      其中<principal_name>表示用户使用的Kerberos用户的principal，如“test@*<系统域名>*”。<path_to_keytab>表示<principal_name>对应的keytab文件路径，如“/opt/auth/test/user.keytab”。

    - 普通模式下：

      ```
      jdbc:hive2://
      <zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;
      ```

      示例：普通模式下通过Beeline客户端连接时执行以下命令：

      sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkN**

ode3_IP>:<zkNode3_Port>/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=sparkthriftserver2x;"

- 非多租户模式

  非多租户模式的客户端连接的是某个指定JDBCServer节点。该模式的连接字符串相比多主实例模式的去掉关于ZooKeeper的参数项“serviceDiscoveryMode”和“zooKeeperNamespace”。

  示例：安全模式下通过Beeline客户端连接非多租户模式时执行以下命令：

  sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://***<server_IP>:<server_Port>*/;user.principal=spark/hadoop.*<系统域名>*@*<系统域名>*;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.*<系统域名>*@*<系统域名>*;"

  说明

  - 其中“<server_IP>:<server_Port>”是指定JDBCServer节点的URL。
  - “CLIENT_HOME”是指客户端路径。

  多租户模式与非多租户模式两种模式的JDBCServer接口相比，除连接方式不同外其他使用方法相同。由于Spark JDBCServer是Hive中的HiveServer2的另外一个实现，其使用方法，请参见Hive官网：**https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**。

  指定租户

  一般情况下，某用户提交的客户端会连接到该用户默认所属租户的JDBCServer上，若需要连接客户端到指定租户的JDBCServer上，可以通过添加**--hiveconf mapreduce.job.queuename**进行指定。

  通过Beeline连接的命令示例如下（aaa为租户名称）：

  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.*<系统域名>*@*<系统域名>*;'

6.30.3 Spark 与其他组件的关系

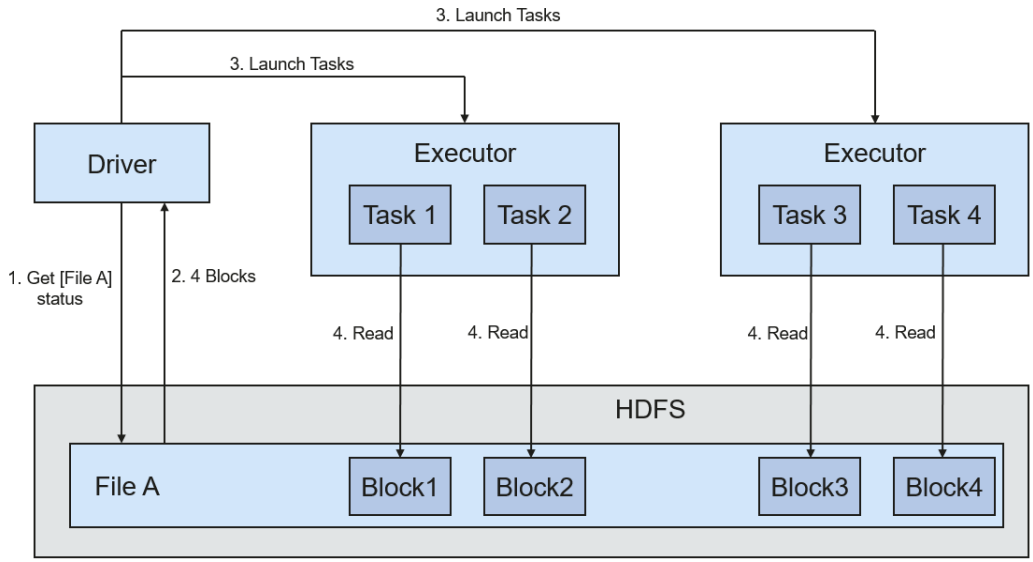
Spark 和 HDFS 的关系

通常，Spark中计算的数据可以来自多个数据源，如Local File、HDFS等。最常用的是HDFS，用户可以一次读取大规模的数据进行并行计算。在计算完成后，也可以将数据存储到HDFS。

分解来看，Spark分成控制端(Driver)和执行端（Executor）。控制端负责任务调度，执行端负责任务执行。

读取文件的过程如图 读取文件过程所示。

图 6-104 读取文件过程

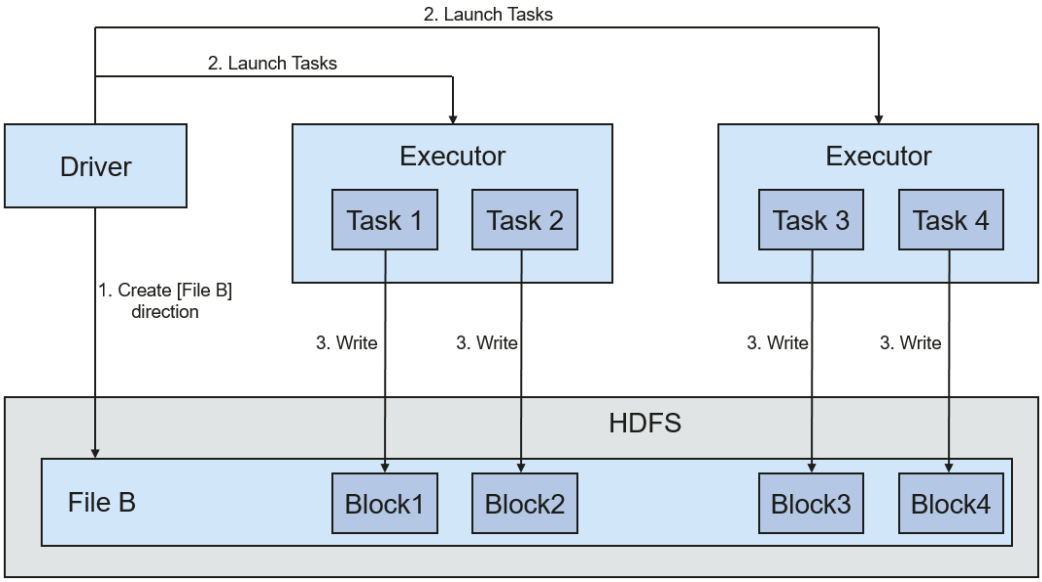


读取文件步骤的详细描述如下所示：

1. Driver与HDFS交互获取File A的文件信息。
2. HDFS返回该文件具体的Block信息。
3. Driver根据具体的Block数据量，决定一个并行度，创建多个Task去读取这些文件Block。
4. 在Executor端执行Task并读取具体的Block，作为RDD(弹性分布数据集)的一部分。

写入文件的过程如图 写入文件过程所示。

图 6-105 写入文件过程



HDFS文件写入的详细步骤如下所示：

1. Driver创建要写入文件的目录。
2. 根据RDD分区分块情况，计算出写数据的Task数，并下发这些任务到Executor。
3. Executor执行这些Task，将具体RDD的数据写入到步骤1创建的目录下。

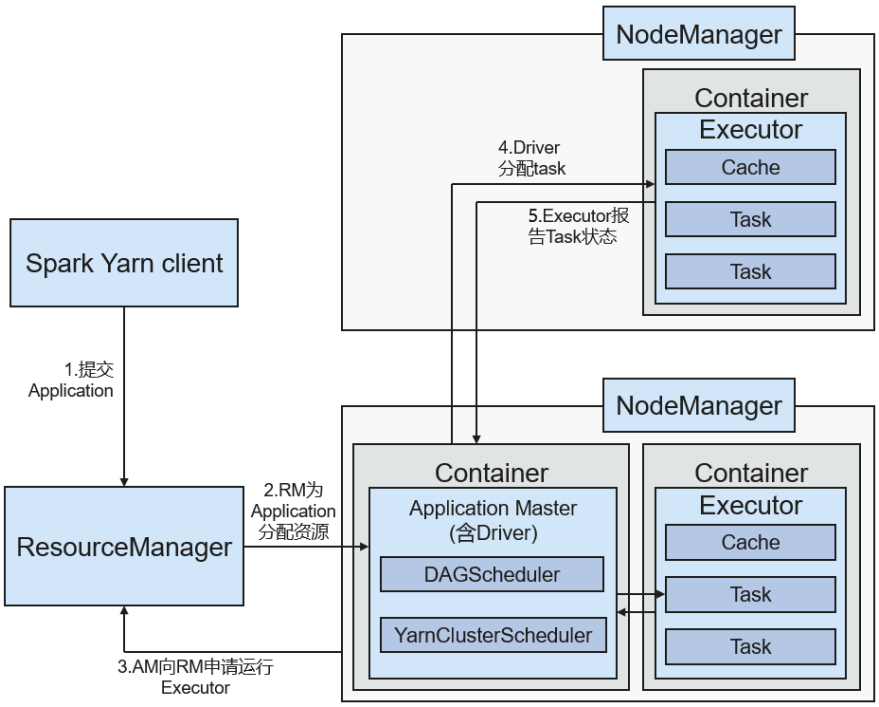
Spark 和 YARN 的关系

Spark的计算调度方式，可以通过YARN的模式实现。Spark共享YARN集群提供丰富的计算资源，将任务分布式的运行起来。Spark on YARN分两种模式：YARN Cluster和YARN Client。

- YARN Cluster模式

  运行框架如图 Spark on yarn-cluster运行框架所示。

  图 6-106 Spark on yarn-cluster 运行框架



  Spark on yarn-cluster实现流程：

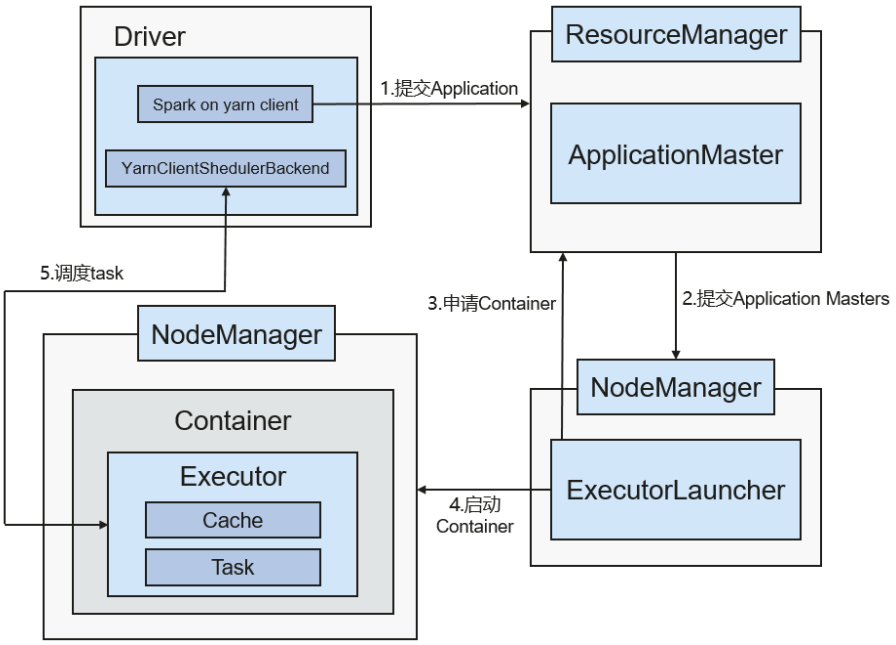
  a. 首先由客户端生成Application信息，提交给ResourceManager。
  b. ResourceManager为Spark Application分配第一个Container(ApplicationMaster)，并在该Container上启动Driver。
  c. ApplicationMaster向ResourceManager申请资源以运行Container。

     ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。
  d. Driver分配Task给Executor执行。
  e. Executor执行Task并向Driver汇报运行状况。

- YARN Client模式

  运行框架如图 Spark on yarn-client运行框架所示。

图 6-107 Spark on yarn-client 运行框架



Spark on yarn-client实现流程：

说明

在yarn-client模式下，Driver部署在Client端，在Client端启动。yarn-client模式下，不兼容老版本的客户端。推荐使用yarn-cluster模式。

a. 客户端向ResourceManager发送Spark应用提交请求，ResourceManager为其返回应答，该应答中包含多种信息(如ApplicationId、可用资源使用上限和下限等)。Client端将启动ApplicationMaster所需的所有信息打包，提交给ResourceManager上。

b. ResourceManager收到请求后，会为ApplicationMaster寻找合适的节点，并在该节点上启动它。ApplicationMaster是Yarn中的角色，在Spark中进程名字是ExecutorLauncher。

c. 根据每个任务的资源需求，ApplicationMaster可向ResourceManager申请一系列用于运行任务的Container。

d. 当ApplicationMaster（从ResourceManager端）收到新分配的Container列表后，会向对应的NodeManager发送信息以启动Container。

   ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。

   说明

   正在运行的container不会被挂起释放资源。

e. Driver分配Task给Executor执行。Executor执行Task并向Driver汇报运行状况。

6.30.4 Spark 开源增强特性

跨源复杂数据的 SQL 查询优化

出于管理和信息收集的需要，企业内部会存储海量数据，包括数目众多的各种数据库、数据仓库等，此时会面临以下困境：数据源种类繁多，数据集结构化混合，相关数据存放分散等，这就导致了跨源复杂查询因传输效率低，耗时长。

当前开源Spark在跨源查询时，只能对简单的filter进行下推，因此造成大量不必要的数据传输，影响SQL引擎性能。针对下推能力进行增强，当前对aggregate、复杂projection、复杂predicate均可以下推到数据源，尽量减少不必要数据的传输，提升查询性能。

目前仅支持JDBC数据源的查询下推，支持的下推模块有aggregate、projection、predicate、aggregate over inner join、aggregate over union all等。为应对不同应用场景的特殊需求，对所有下推模块设计开关功能，用户可以自行配置是否应用上述查询下推的增强。

表 6-24 跨源查询增加特性对比

模块	增强前	增强后
aggregate	不支持aggregate下推	• 支持的聚合函数为：sum, avg, max, min, count 例如：select count(*) from table • 支持聚合函数内部表达式 例如：select sum(a+b) from table • 支持聚合函数运算，例如：select avg(a) + max(b) from table • 支持having下推 例如：select sum(a) from table where a>0 group by b having sum(a)>10 • 支持部分函数下推 支持对abs()、month()、length()等数学、时间、字符串函数进行下推。并且，除了以上内置函数，用户还可以通过SET命令新增数据源支持的函数。 例如：select sum(abs(a)) from table • 支持aggregate之后的limit、order by下推（由于Oracle不支持limit，所以Oracle中limit、order by不会下推） 例如：select sum(a) from table where a>0 group by b order by sum(a) limit 5

模块	增强前	增强后
projection	仅支持简单projection下推，例如： select a, b from table	• 支持复杂表达式下推。 例如：select (a+b)*c from table • 支持部分函数下推，详细参见表下方的说明。 例如：select length(a)+abs(b) from table • 支持projection之后的limit、order by下推。 例如：select a, b+c from table order by a limit 3
predicate	仅支持运算符左边为列名右边为值的简单filter，例如 select * from table where a>0 or b in (“aaa”, “bbb”)	• 支持复杂表达数下推 例如：select * from table where a +b>c*d or a/c in (1, 2, 3) • 支持部分函数下推，详细参见表下方的说明。 例如：select * from table where length(a)>5
aggregate over inner join	需要将两个表中相关的数据全部加载到Spark，先进行join操作，再进行aggregate操作	支持以下几种： • 支持的聚合函数为：sum, avg, max, min,count • 所有aggregate只能来自同一个表，group by可以来自一个表或者两个表，只支持inner join。 不支持的情形有： • 不支持aggregate同时来自join左表和右表的下推。 • 不支持aggregate内包含运算，如：sum(a+b)。 • 不支持aggregate运算，如：sum(a) +min(b)。
aggregate over union all	需要将两个表中相关的数据全部加载到Spark，先进行union操作，再进行aggregate操作	支持情况： 支持的聚合函数为：sum, avg, max, min,count 不支持的情况： • 不支持aggregate内包含运算，如：sum(a+b)。 • 不支持aggregate运算，如：sum(a) +min(b)。

注意事项

- 外部数据源是Hive的场景，通过Spark建的外表无法进行查询。
- 数据源只支持MySQL和MPPDB。

6.31 Spark2x

6.31.1 Spark2x 基本原理

简介

Spark是基于内存的分布式计算框架。在迭代计算的场景下，数据处理过程中的数据可以存储在内存中，提供了比MapReduce高10到100倍的计算能力。Spark可以使用HDFS作为底层存储，使用户能够快速地从MapReduce切换到Spark计算平台上去。Spark提供一站式数据分析能力，包括小批量流式处理、离线批处理、SQL查询、数据挖掘等，用户可以在同一个应用中无缝结合使用这些能力。Spark2x的开源新特性请参考Spark2x开源新特性说明。

更多关于Spark2x组件操作指导，请参考使用Spark/Spark2x。

Spark的特点如下：

- 通过分布式内存计算和DAG（无回路有向图）执行引擎提升数据处理能力，比MapReduce性能高10倍到100倍。
- 提供多种语言开发接口（Scala/Java/Python），并且提供几十种高度抽象算子，可以很方便构建分布式的数据处理应用。
- 结合SQL、Streaming等形成数据处理栈，提供一站式数据处理能力。
- 支持契合Hadoop生态环境，Spark应用可以运行在Standalone、Mesos或者YARN上，能够接入HDFS、HBase、Hive等多种数据源，支持MapReduce程序平滑转接。

结构

Spark的架构如图6-108所示，各模块的说明如表6-25所示。

图 6-108 Spark 架构

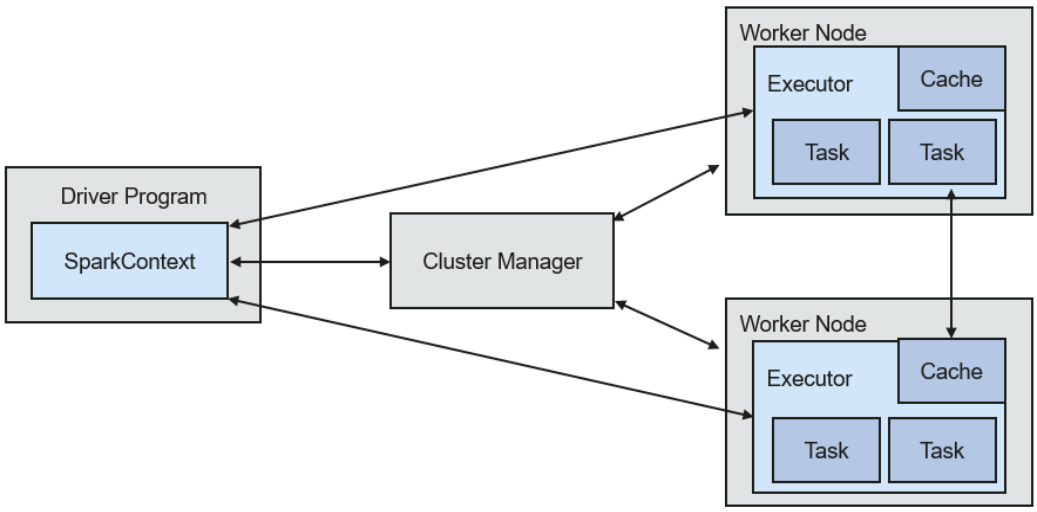


表 6-25 基本概念说明

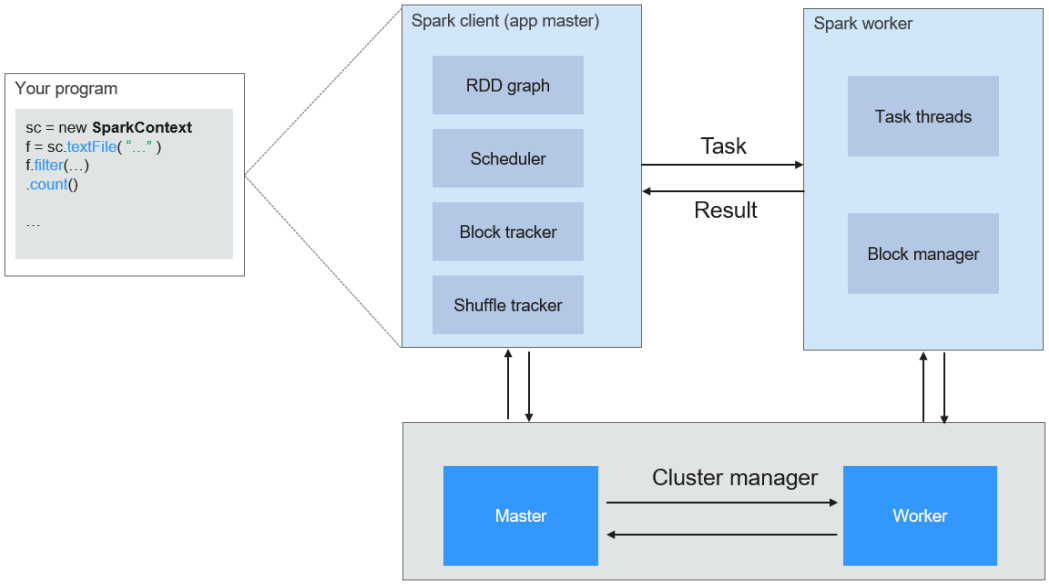
模块	说明
Cluster Manager	集群管理器，管理集群中的资源。Spark支持多种集群管理器，Spark自带的Standalone集群管理器、Mesos或YARN。Spark集群默认采用YARN模式。
Application	Spark应用，由一个Driver Program和多个Executor组成。
Deploy Mode	部署模式，分为cluster和client模式。cluster模式下，Driver会在集群内的节点运行；而在client模式下，Driver在客户端运行（集群外）。
Driver Program	是Spark应用程序的主进程，运行Application的main()函数并创建SparkContext。负责应用程序的解析、生成Stage并调度Task到Executor上。通常SparkContext代表Driver Program。
Executor	在Work Node上启动的进程，用来执行Task，管理并处理应用中使用到的数据。一个Spark应用一般包含多个Executor，每个Executor接收Driver的命令，并执行一到多个Task。
Worker Node	集群中负责启动并管理Executor以及资源的节点。
Job	一个Action算子（比如collect算子）对应一个Job，由并行计算的多个Task组成。
Stage	每个Job由多个Stage组成，每个Stage是一个Task集合，由DAG分割而成。
Task	承载业务逻辑的运算单元，是Spark平台上可执行的最小工作单元。一个应用根据执行计划以及计算量分为多个Task。

Spark 原理

Spark的应用运行架构如图6-109所示，运行流程如下所示：

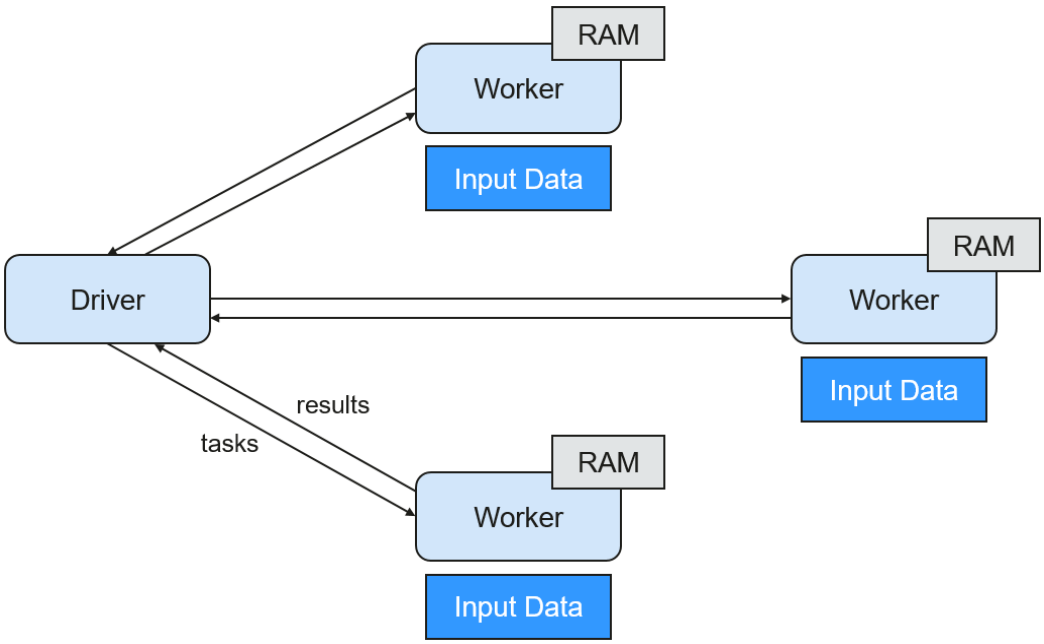
1. 应用程序（Application）是作为一个进程的集合运行在集群上的，由Driver进行协调。
2. 在运行一个应用时，Driver会去连接集群管理器（Standalone、Mesos、YARN）申请运行Executor资源，并启动ExecutorBackend。然后由集群管理器在不同的应用之间调度资源。Driver同时会启动应用程序DAG调度、Stage划分、Task生成。
3. 然后Spark会把应用的代码（传递给SparkContext的JAR或者Python定义的代码）发送到Executor上。
4. 所有的Task执行完成后，用户的应用程序运行结束。

图 6-109 Spark 应用运行架构



Spark采用Master和Worker的模式，如图6-110所示。用户在Spark客户端提交应用程序，调度器将Job分解为多个Task发送到各个Worker中执行，各个Worker将计算的结果上报给Driver（即Master），Driver聚合结果返回给客户端。

图 6-110 Spark 的 Master 和 Worker



在此结构中，有几个说明点：

- 应用之间是独立的。

  每个应用有自己的executor进程，Executor启动多个线程，并行地执行任务。无论是在调度方面，或者是executor方面。各个Driver独立调度自己的任务；不同的应用任务运行在不同的JVM上，即不同的Executor。

- 不同Spark应用之间是不共享数据的，除非把数据存储在外部的存储系统上（比如HDFS）。
- 因为Driver程序在集群上调度任务，所以Driver程序需要和worker节点比较近，比如在一个相同的局部网络内。

Spark on YARN有两种部署模式：

- YARN-Cluster模式下，Spark的Driver会运行在YARN集群内的ApplicationMaster进程中，ApplicationMaster已经启动之后，提交任务的客户端退出也不会影响任务的运行。
- YARN-Client模式下，Driver启动在客户端进程内，ApplicationMaster进程只用来向YARN集群申请资源。

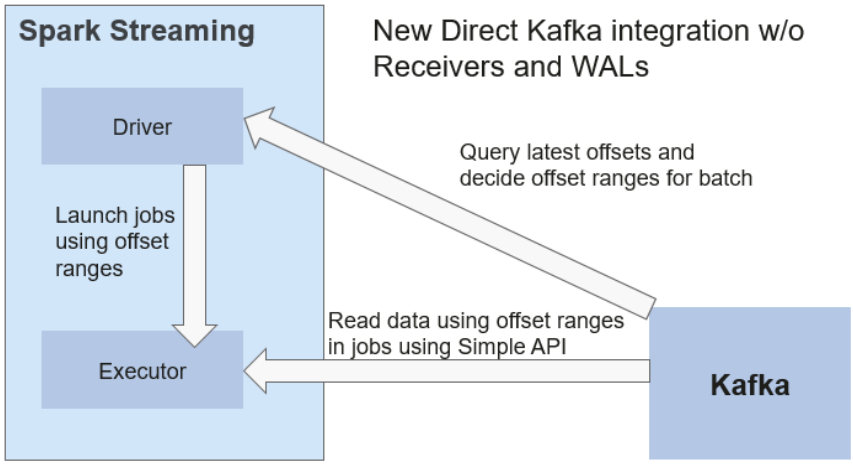
Spark Streaming 原理

Spark Streaming是一种构建在Spark上的实时计算框架，扩展了Spark处理大规模流式数据的能力。当前Spark支持两种数据处理方式：Direct Streaming和Receiver方式。

Direct Streaming计算流程

Direct Streaming方式主要通过采用Direct API对数据进行处理。以Kafka Direct接口为例，与启动一个Receiver来连续不断地从Kafka中接收数据并写入到WAL中相比，Direct API简单地给出每个batch区间需要读取的偏移量位置。然后，每个batch的Job被运行，而对应偏移量的数据在Kafka中已准备好。这些偏移量信息也被可靠地存储在checkpoint文件中，应用失败重启时可以直接读取偏移量信息。

图 6-111 Direct Kafka 接口数据传输



需要注意的是，Spark Streaming可以在失败后重新从Kafka中读取并处理数据段。然而，由于语义仅被处理一次，重新处理的结果和没有失败处理的结果是一致的。

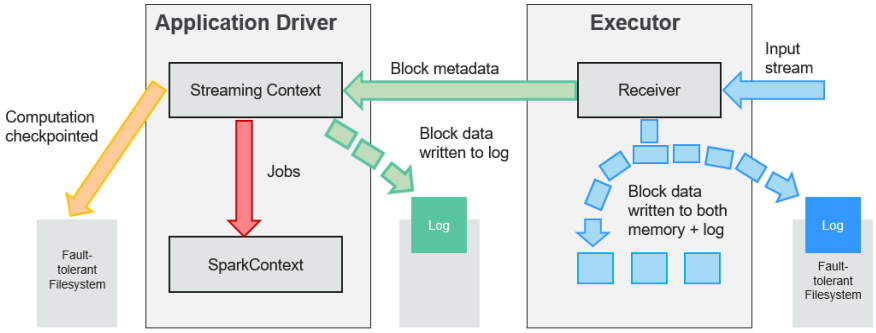
因此，Direct API消除了需要使用WAL和Receivers的情况，且确保每个Kafka记录仅被接收一次，这种接收更加高效。使得Spark Streaming和Kafka可以很好地整合在一起。总体来说，这些特性使得流处理管道拥有高容错性、高效性及易用性，因此推荐使用Direct Streaming方式处理数据。

Receiver计算流程

在一个Spark Streaming应用开始时（也就是Driver开始时），相关的StreamingContext（所有流功能的基础）使用SparkContext启动Receiver成为长驻运

行任务。这些Receiver接收并保存流数据到Spark内存中以供处理。用户传送数据的生命周期如图6-112所示：

图 6-112 数据传输生命周期



1. 接收数据（蓝色箭头）

   Receiver将数据流分成一系列小块，存储到Executor内存中。另外，在启用预写日志（Write-ahead Log，简称WAL）以后，数据同时还写入到容错文件系统的预写日志中。

2. 通知Driver（绿色箭头）

   接收块中的元数据（Metadata）被发送到Driver的StreamingContext。这个元数据包括：

   - 定位其在Executor内存中数据位置的块Reference ID。
   - 若启用了WAL，还包括块数据在日志中的偏移信息。

3. 处理数据（红色箭头）

   对每个批次的数据，StreamingContext使用Block信息产生RDD及其Job。StreamingContext通过运行任务处理Executor内存中的Block来执行Job。

4. 周期性地设置检查点（橙色箭头）

5. 为了容错的需要，StreamingContext会周期性地设置检查点，并保存到外部文件系统中。

容错性

Spark及其RDD允许无缝地处理集群中任何Worker节点的故障。鉴于Spark Streaming建立于Spark之上，因此其Worker节点也具备了同样的容错能力。然而，由于Spark Streaming的长正常运行需求，其应用程序必须也具备从Driver进程（协调各个Worker的主要应用进程）故障中恢复的能力。使Spark Driver能够容错是件很棘手的事情，因为可能是任意计算模式实现的任意用户程序。不过Spark Streaming应用程序在计算上有一个内在的结构：在每批次数据周期性地执行同样的Spark计算。这种结构允许把应用的状态（亦称Checkpoint）周期性地保存到可靠的存储空间中，并在Driver重新启动时恢复该状态。

对于文件这样的源数据，这个Driver恢复机制足以做到零数据丢失，因为所有的数据都保存在了像HDFS这样的容错文件系统中。但对于像Kafka和Flume等其他数据源，有些接收到的数据还只缓存在内存中，尚未被处理，就有可能会丢失。这是由于Spark应用的分布操作方式引起的。当Driver进程失败时，所有在Cluster Manager中运行的Executor，连同在内存中的所有数据，也同时被终止。为了避免这种数据损失，Spark Streaming引进了WAL功能。

WAL通常被用于数据库和文件系统中，用来保证任何数据操作的持久性，即先将操作记入一个持久的日志，再对数据施加这个操作。若施加操作的过程中执行失败了，则

通过读取日志并重新施加前面指定的操作，系统就得到了恢复。下面介绍了如何利用这样的概念保证接收到的数据的持久性。

Kafka数据源使用Receiver来接收数据，是Executor中的长运行任务，负责从数据源接收数据，并且在数据源支持时还负责确认收到数据的结果（收到的数据被保存在Executor的内存中，然后Driver在Executor中运行来处理任务）。

当启用了预写日志以后，所有收到的数据同时还保存到了容错文件系统的日志文件中。此时即使Spark Streaming失败，这些接收到的数据也不会丢失。另外，接收数据的正确性只在数据被预写到日志以后Receiver才会确认，已经缓存但还没有保存的数据可以在Driver重新启动之后由数据源再发送一次。这两个机制确保了零数据丢失，即所有的数据或者从日志中恢复，或者由数据源重发。

如果需要启用预写日志功能，可以通过如下动作实现：

- 通过“streamingContext.checkpoint”(path-to-directory)设置checkpoint的目录，这个目录是一个HDFS的文件路径，既用作保存流的checkpoint，又用作保存预写日志。
- 设置SparkConf的属性“spark.streaming.receiver.writeAheadLog.enable”为“true”（默认值是“false”）。

在WAL被启用以后，所有Receiver都获得了能够从可靠收到的数据中恢复的优势。建议缓存RDD时不采取多备份选项，因为用于预写日志的容错文件系统很可能也复制了数据。

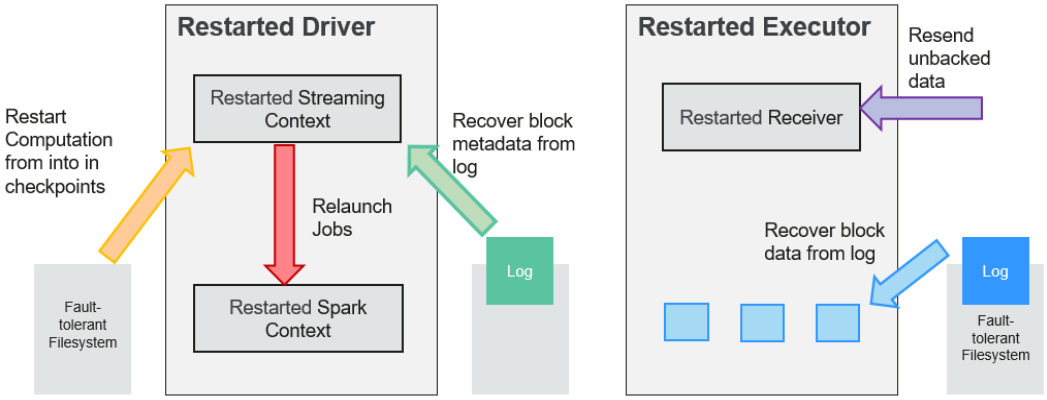
说明

在启用了预写日志以后，数据接收吞吐率会有降低。由于所有数据都被写入容错文件系统，文件系统的写入吞吐率和用于数据复制的网络带宽，可能就是潜在的瓶颈了。在此情况下，需要创建更多的Receiver增加数据接收的并行度，或使用更好的硬件以增加容错文件系统的吞吐率。

恢复流程

当一个失败的Driver重启时，按如下流程启动：

图 6-113 计算恢复流程



1. 恢复计算（橙色箭头）
   使用checkpoint信息重启Driver，重新构造SparkContext并重启Receiver。
2. 恢复元数据块（绿色箭头）
   为了保证能够继续下去所必备的全部元数据块都被恢复。

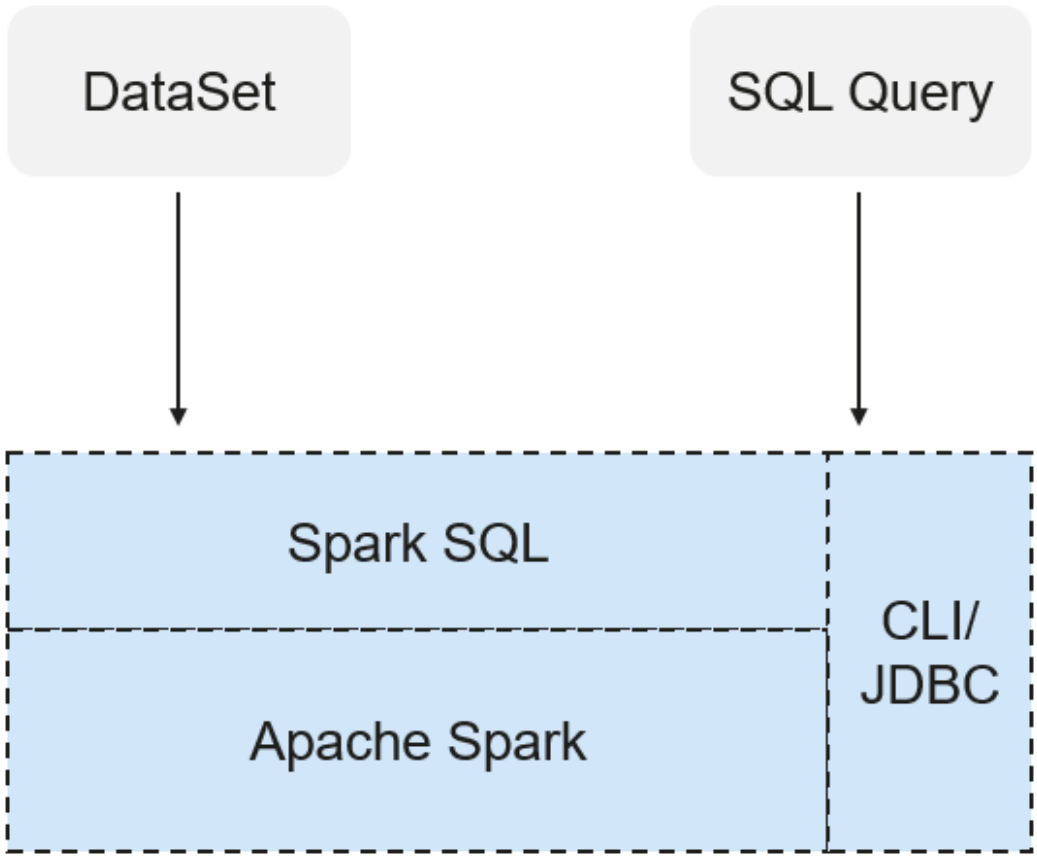
3. 未完成作业的重新形成（红色箭头）
   由于失败而没有处理完成的批处理，将使用恢复的元数据再次产生RDD和对应的作业。
4. 读取保存在日志中的块数据（蓝色箭头）
   在这些作业执行时，块数据直接从预写日志中读出。这将恢复在日志中可靠地保存的所有必要数据。
5. 重发尚未确认的数据（紫色箭头）
   失败时没有保存到日志中的缓存数据将由数据源再次发送。因为Receiver尚未对其确认。

因此通过预写日志和可靠的Receiver，Spark Streaming就可以保证没有输入数据会由于Driver的失败而丢失。

SparkSQL 和 DataSet 原理

SparkSQL

图 6-114 SparkSQL 和 DataSet



Spark SQL是Spark中用于结构化数据处理的模块。在Spark应用中，可以无缝地使用SQL语句亦或是DataSet API对结构化数据进行查询。

Spark SQL以及DataSet还提供了一种通用的访问多数据源的方式，可访问的数据源包括Hive、CSV、Parquet、ORC、JSON和JDBC数据源，这些不同的数据源之间也可以实现互相操作。Spark SQL复用了Hive的前端处理逻辑和元数据处理模块，使用Spark SQL可以直接对已有的Hive数据进行查询。

另外，SparkSQL还提供了诸如API、CLI、JDBC等诸多接口，对客户端提供多样接入形式。

Spark SQL Native DDL/DML

Spark1.5将很多DDL/DML命令下压到Hive执行，造成了与Hive的耦合，且在一定程度上不够灵活（比如报错不符合预期、结果与预期不一致等）。

Spark2x实现了命令的本地化，使用Spark SQL Native DDL/DML取代Hive执行DDL/DML命令。一方面实现和Hive的解耦，另一方面可以对命令进行定制化。

DataSet

DataSet是一个由特定域的对象组成的强类型集合，可通过功能或关系操作并行转换其中的对象。 每个Dataset还有一个非类型视图，即由多个列组成的DataSet，称为DataFrame。

DataFrame是一个由多个列组成的结构化的分布式数据集合，等同于关系数据库中的一张表，或者是R/Python中的data frame。DataFrame是Spark SQL中的最基本的概念，可以通过多种方式创建，例如结构化的数据集、Hive表、外部数据库或者是RDD。

可用于DataSet的操作分为Transformation和Action。

- Transformation操作可生成新的DataSet。
  如map、filter、select和aggregate (groupBy)。
- Action操作可触发计算及返回记结果。
  如count、show或向文件系统写数据。

通常使用两种方法创建一个DataSet：

- 最常见的方法是通过使用SparkSession上的read函数将Spark指向存储系统上的某些文件。
  ```
  val people = spark.read.parquet("...").as[Person]  // Scala
  DataSet<Person> people = spark.read().parquet("...").as(Encoders.bean(Person.class));//Java
  ```
- 还可通过已存在的DataSet上可用的transformation操作来创建数据集。 例如，在已存在的DataSet上应用map操作来创建新的DataSet：
  ```
  val names = people.map(_.name)  // 使用Scala语言，且names为一个Dataset
  Dataset<String> names = people.map((Person p) -> p.name, Encoders.STRING)); // Java
  ```

CLI和JDBCServer

除了API编程接口之外，Spark SQL还对外提供CLI/JDBC接口：

- spark-shell和spark-sql脚本均可以提供CLI，以便于调试。
- JDBCServer提供JDBC接口，外部可直接通过发送JDBC请求来完成结构化数据的计算和解析。

SparkSession 原理

SparkSession是Spark2x编程的统一API，也可看作是读取数据的统一入口。SparkSession提供了一个统一的入口点来执行以前分散在多个类中的许多操作，并且还为那些较旧的类提供了访问器方法，以实现最大的兼容性。

使用构建器模式创建SparkSession。如果存在SparkSession，构建器将自动重用现有的SparkSession；如果不存在则会创建一个SparkSession。 在I/O期间，在构建器中设置的配置项将自动同步到Spark和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()
```

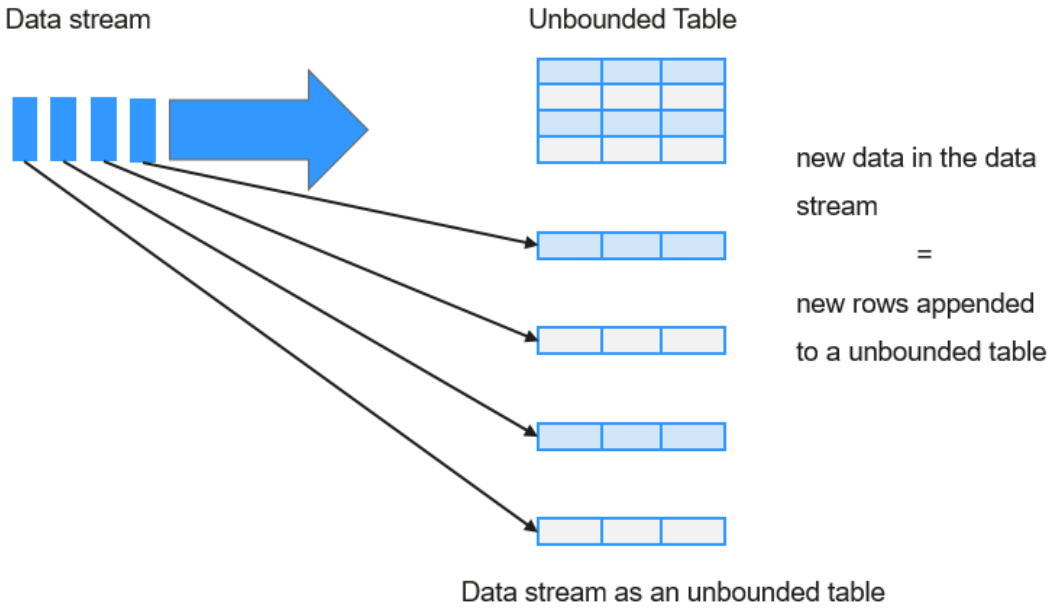
- SparkSession可以用于对数据执行SQL查询，将结果返回为DataFrame。
  ```
  sparkSession.sql("select * from person").show
  ```
- SparkSession可以用于设置运行时的配置项，这些配置项可以在SQL中使用变量替换。
  ```
  sparkSession.conf.set("spark.some.config", "abcd")
  sparkSession.conf.get("spark.some.config")
  sparkSession.sql("select ${spark.some.config}")
  ```
- SparkSession包括一个“catalog”方法，其中包含使用Metastore（即数据目录）的方法。方法返回值为数据集，可以使用相同的Dataset API来运行。
  ```
  val tables = sparkSession.catalog.listTables()
  val columns = sparkSession.catalog.listColumns("myTable")
  ```
- 底层SparkContext可以通过SparkSession的SparkContext API访问。
  ```
  val sparkContext = sparkSession.sparkContext
  ```

Structured Streaming 原理

Structured Streaming是构建在Spark SQL引擎上的流式数据处理引擎，用户可以使用Scala、Java、Python或R中的Dataset/DataFrame API进行流数据聚合运算、按事件时间窗口计算、流流Join等操作。当流数据连续不断地产生时，Spark SQL将会增量的、持续不断地处理这些数据并将结果更新到结果集中。同时，系统通过checkpoint和Write Ahead Logs确保端到端的完全一次性容错保证。

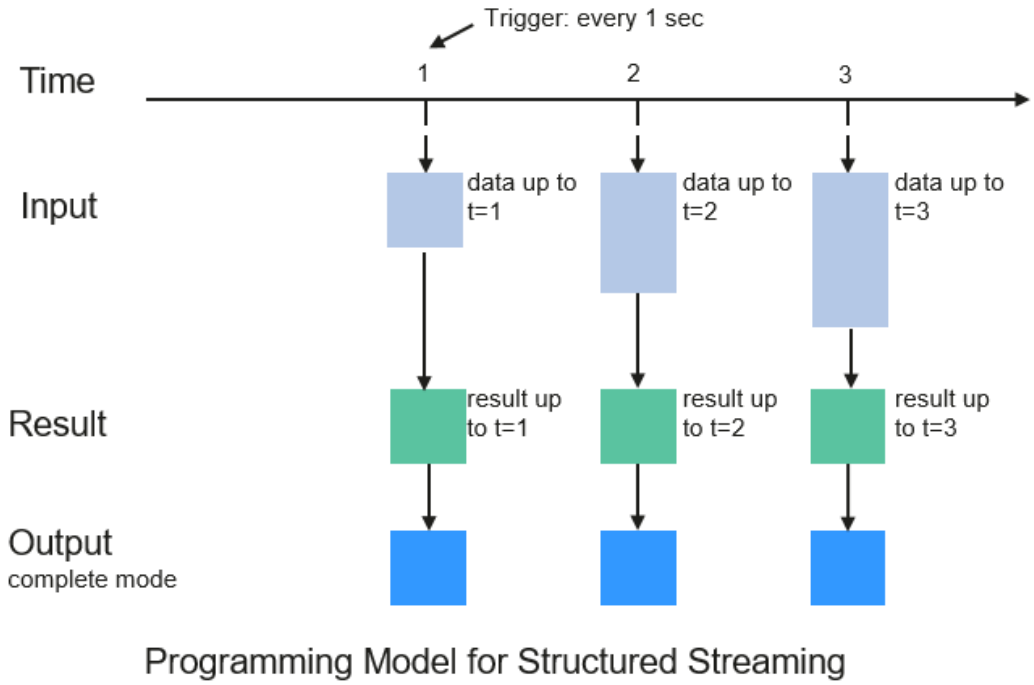
Structured Streaming的核心是将流式的数据看成一张不断增加的数据库表，这种流式的数据处理模型类似于数据块处理模型，可以把静态数据库表的一些查询操作应用在流式计算中，Spark执行标准的SQL查询，从不断增加的无边界表中获取数据。

图 6-115 Structured Streaming 无边界表



每一条查询的操作都会产生一个结果集Result Table。每一个触发间隔，当新的数据新增到表中，都会最终更新Result Table。无论何时结果集发生了更新，都能将变化的结果写入一个外部的存储系统。

图 6-116 Structured Streaming 数据处理模型



Structured Streaming在OutPut阶段可以定义不同的存储方式，有如下3种：

- Complete Mode：整个更新的结果集都会写入外部存储。整张表的写入操作将由外部存储系统的连接器完成。
- Append Mode：当时间间隔触发时，只有在Result Table中新增加的数据行会被写入外部存储。这种方式只适用于结果集中已经存在的内容不希望发生改变的情况下，如果已经存在的数据会被更新，不适合使用此种方式。
- Update Mode：当时间间隔触发时，只有在Result Table中被更新的数据才会被写入外部存储系统。注意，和Complete Mode方式的不同之处是不更新的结果集不会写入外部存储。

基本概念

- **RDD**

  即弹性分布数据集（Resilient Distributed Dataset），是Spark的核心概念。指的是一个只读的，可分区的分布式数据集，这个数据集的全部或部分可以缓存在内存中，在多次计算间重用。

  RDD的生成：

  - 从HDFS输入创建，或从与Hadoop兼容的其他存储系统中输入创建。
  - 从父RDD转换得到新RDD。
  - 从数据集合转换而来，通过编码实现。

  RDD的存储：

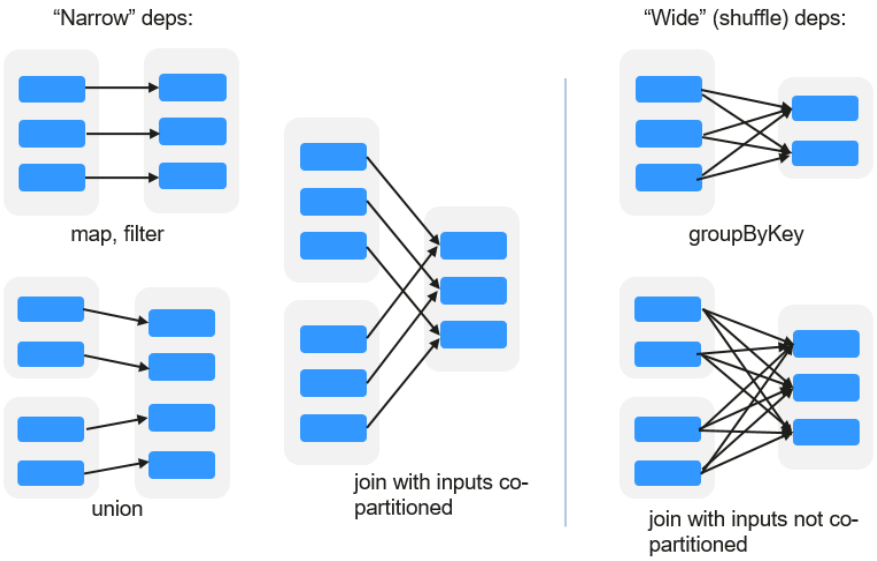
  - 用户可以选择不同的存储级别缓存RDD以便重用（RDD有11种存储级别）。

- 当前RDD默认是存储于内存，但当内存不足时，RDD会溢出到磁盘中。

- **Dependency（RDD的依赖）**

  RDD的依赖分别为：窄依赖和宽依赖。

  图 6-117 RDD 的依赖

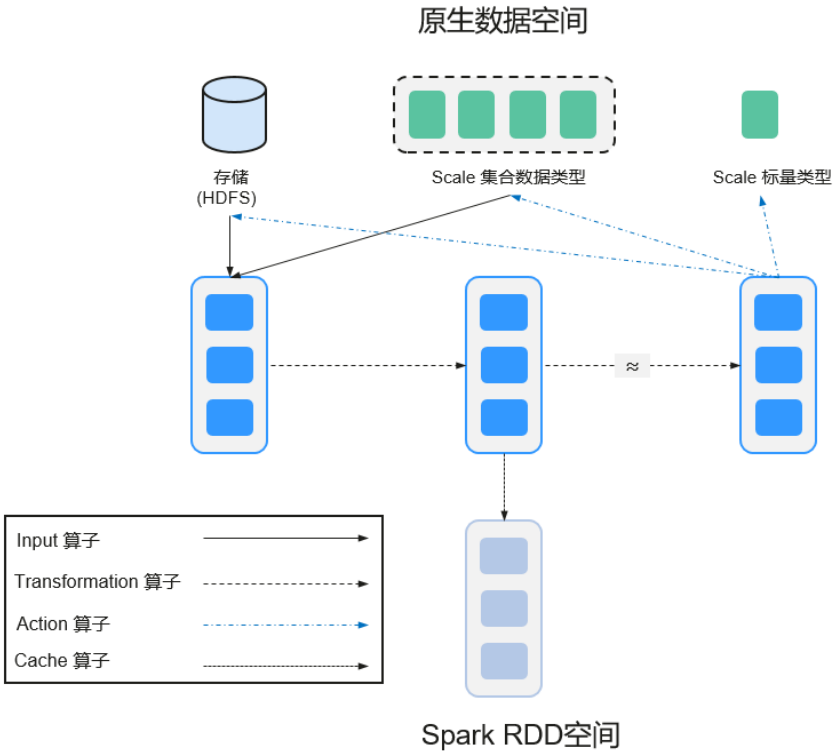
  - **窄依赖：**指父RDD的每一个分区最多被一个子RDD的分区所用。
  - **宽依赖：**指子RDD的分区依赖于父RDD的所有分区。

  窄依赖对优化很有利。逻辑上，每个RDD的算子都是一个fork/join（此join非上文的join算子，而是指同步多个并行任务的barrier）：把计算fork到每个分区，算完后join，然后fork/join下一个RDD的算子。如果直接翻译到物理实现，是很不经济的：一是每一个RDD（即使是中间结果）都需要物化到内存或存储中，费时费空间；二是join作为全局的barrier，是很昂贵的，会被最慢的那个节点拖死。如果子RDD的分区到父RDD的分区是窄依赖，就可以实施经典的fusion优化，把两个fork/join合为一个；如果连续的变换算子序列都是窄依赖，就可以把很多个fork/join并为一个，不但减少了大量的全局barrier，而且无需物化很多中间结果RDD，这将极大地提升性能。Spark把这个叫做流水线（pipeline）优化。

- **Transformation和Action（RDD的操作）**

  对RDD的操作包含Transformation（返回值还是一个RDD）和Action（返回值不是一个RDD）两种。RDD的操作流程如图6-118所示。其中Transformation操作是Lazy的，也就是说从一个RDD转换生成另一个RDD的操作不是马上执行，Spark在遇到Transformations操作时只会记录需要这样的操作，并不会去执行，需要等到有Actions操作的时候才会真正启动计算过程进行计算。Actions操作会返回结果或把RDD数据写到存储系统中。Actions是触发Spark启动计算的动因。

图 6-118 RDD 操作示例



RDD看起来与Scala集合类型没有太大差别，但数据和运行模型大相迥异。

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

a. textFile算子从HDFS读取日志文件，返回file（作为RDD）。

b. filter算子筛出带“ERROR”的行，赋给errors（新RDD）。filter算子是一个Transformation操作。

c. cache算子缓存下来以备未来使用。

d. count算子返回errors的行数。count算子是一个Action操作。

Transformation操作可以分为如下几种类型：

- 视RDD的元素为简单元素。

  输入输出一对一，且结果RDD的分区结构不变，主要是map。

  输入输出一对多，且结果RDD的分区结构不变，如flatMap（map后由一个元素变为一个包含多个元素的序列，然后展平为一个个的元素）。

  输入输出一对一，但结果RDD的分区结构发生了变化，如union（两个RDD合为一个，分区数变为两个RDD分区数之和）、coalesce（分区减少）。

  从输入中选择部分元素的算子，如filter、distinct（去除重复元素）、subtract（本RDD有、其他RDD无的元素留下来）和sample（采样）。

- 视RDD的元素为Key-Value对。

  对单个RDD做一对一运算，如mapValues（保持源RDD的分区方式，这与map不同）；

  对单个RDD重排，如sort、partitionBy（实现一致性的分区划分，这个对数据本地性优化很重要）；

对单个RDD基于key进行重组和reduce，如groupByKey、reduceByKey；

对两个RDD基于key进行join和重组，如join、cogroup。

说明

后三种操作都涉及重排，称为shuffle类操作。

Action操作可以分为如下几种：

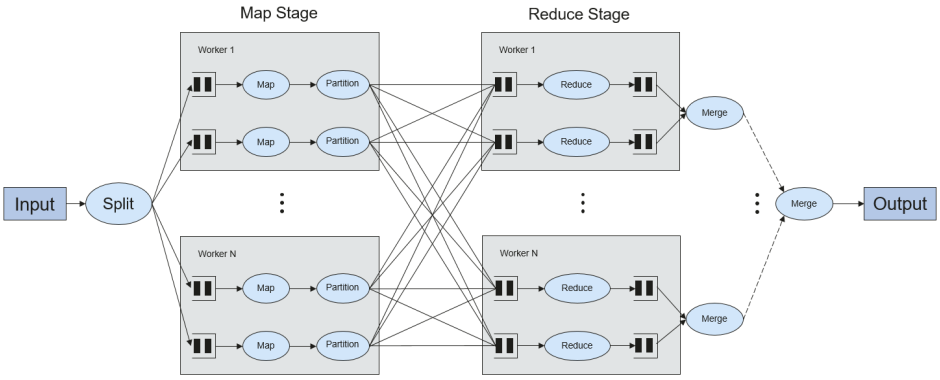
- 生成标量，如count（返回RDD中元素的个数）、reduce、fold/aggregate（返回几个标量）、take（返回前几个元素）。
- 生成Scala集合类型，如collect（把RDD中的所有元素导入Scala集合类型）、lookup（查找对应key的所有值）。
- 写入存储，如与前文textFile对应的saveAsTextFile。
- 还有一个检查点算子checkpoint。当Lineage特别长时（这在图计算中时常发生），出错时重新执行整个序列要很长时间，可以主动调用checkpoint把当前数据写入稳定存储，作为检查点。

- **Shuffle**

  Shuffle是MapReduce框架中的一个特定的phase，介于Map phase和Reduce phase之间，当Map的输出结果要被Reduce使用时，每一条输出结果需要按key哈希，并且分发到对应的Reducer上去，这个过程就是shuffle。由于shuffle涉及到了磁盘的读写和网络的传输，因此shuffle性能的高低直接影响到了整个程序的运行效率。

  下图清晰地描述了MapReduce算法的整个流程。

  图 6-119 算法流程



  概念上shuffle就是一个沟通数据连接的桥梁，实际上shuffle这一部分是如何实现的呢，下面就以Spark为例讲解shuffle在Spark中的实现。

  Shuffle操作将一个Spark的Job分成多个Stage，前面的stages会包括一个或多个ShuffleMapTasks，最后一个stage会包括一个或多个ResultTask。

- **Spark Application的结构**

  Spark Application的结构可分为两部分：初始化SparkContext和主体程序。

  - 初始化SparkContext：构建Spark Application的运行环境。

    构建SparkContext对象，如：

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

    参数介绍：

    master：连接字符串，连接方式有local、yarn-cluster、yarn-client等。

appName：构建的Application名称。

SparkHome：集群中安装Spark的目录。

jars：应用程序代码和依赖包。

– 主体程序：处理数据

提交Application的描述请参见：**https://archive.apache.org/dist/spark/docs/3.1.1/submitting-applications.html**

- **Spark shell命令**

  Spark基本shell命令，支持提交Spark应用。命令为：

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

  参数解释：

  --class：Spark应用的类名。

  --master：Spark用于所连接的master，如yarn-client，yarn-cluster等。

  application-jar：Spark应用的jar包的路径。

  application-arguments：提交Spark应用的所需要的参数（可以为空）。

- **Spark JobHistory Server**

  用于监控正在运行的或者历史的Spark作业在Spark框架各个阶段的细节以及提供日志显示，帮助用户更细粒度地去开发、配置和调优作业。

6.31.2 Spark2x 多主实例

背景介绍

基于社区已有的JDBCServer基础上，采用多主实例模式实现了其高可用性方案。集群中支持同时共存多个JDBCServer服务，通过客户端可以随机连接其中的任意一个服务进行业务操作。即使集群中一个或多个JDBCServer服务停止工作，也不影响用户通过同一个客户端接口连接其他正常的JDBCServer服务。

多主实例模式相比主备模式的HA方案，优势主要体现在对以下两种场景的改进。

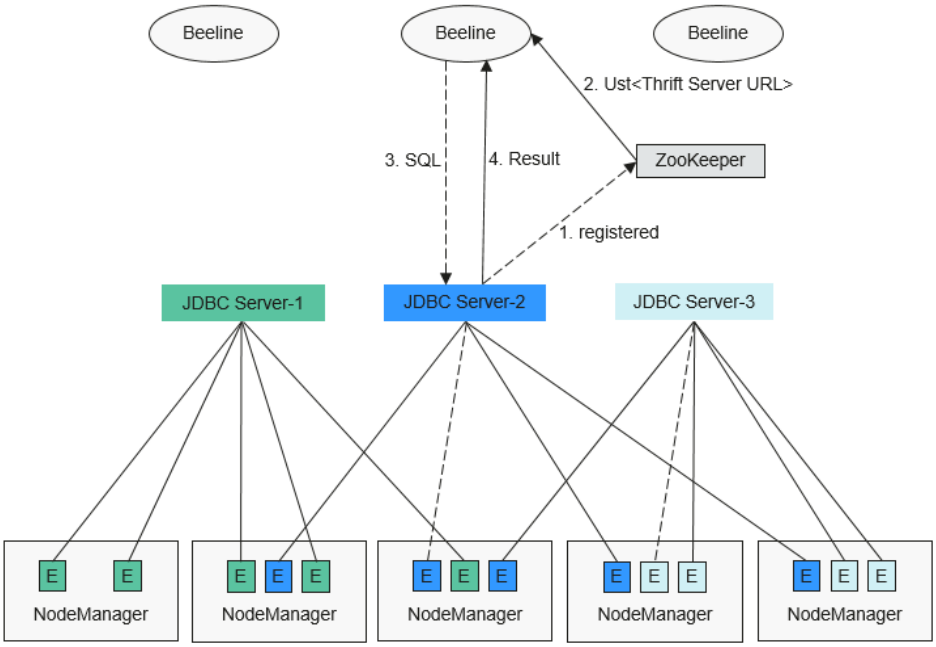
- 主备模式下，当发生主备切换时，会存在一段时间内服务不可用，该段时间JDBCServer无法控制，取决于Yarn服务的资源情况。
- Spark中通过类似于HiveServer2的Thrift JDBC提供服务，用户通过Beeline以及JDBC接口访问。因此JDBCServer集群的处理能力取决于主Server的单点能力，可扩展性不够。

采用多主实例模式的HA方案，不仅可以规避主备切换服务中断的问题，实现服务不中断或少中断，还可以通过横向扩展集群来提高并发能力。

实现方案

多主实例模式的HA方案原理如下图所示。

图 6-120 Spark JDBCServer HA



1. JDBCServer在启动时，向ZooKeeper注册自身消息，在指定目录中写入节点，节点包含了该实例对应的IP，端口，版本号和序列号等信息（多节点信息之间以逗号隔开）。

   示例如下：

   ```
   [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=000
   0001242,serverUri=192.168.81.37:22550;version=8.1.0.1;sequence=0000001243]
   ```

2. 客户端连接JDBCServer时，需要指定Namespace，即访问ZooKeeper哪个目录下的JDBCServer实例。在连接的时候，会从Namespace下随机选择一个实例连接，详细URL参见**URL连接介绍**。
3. 客户端成功连接JDBCServer服务后，向JDBCServer服务发送SQL语句。
4. JDBCServer服务执行客户端发送的SQL语句后，将结果返回给客户端。

在HA方案中，每个JDBCServer服务（即实例）都是独立且等同的，当其中一个实例在升级或者业务中断时，其他的实例也能接受客户端的连接请求。

多主实例方案遵循以下规则：

- 当一个实例异常退出时，其他实例不会接管此实例上的会话，也不会接管此实例上运行的业务。
- 当JDBCServer进程停止时，删除在ZooKeeper上的相应节点。
- 由于客户端选择服务端的策略是随机的，可能会出现会话随机分配不均匀的情况，进而可能引起实例间的负载不均衡。
- 实例进入维护模式（即进入此模式后不再接受新的客户端连接）后，当达到退服超时时间，仍在此实例上运行的业务有可能会发生失败。

URL 连接介绍

多主实例模式

多主实例模式的客户端读取ZooKeeper节点中的内容，连接对应的JDBCServer服务。连接字符串为：

- 安全模式下：
  - Kinit认证方式下的JDBCURL如下所示：
    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.*<系统域名>*@*<系统域名>*;

    说明

    - 其中“<zkNode_IP>:<zkNode_Port>”是ZooKeeper的URL，多个URL以逗号隔开。

      例如：“192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181”。
    - 其中“sparkthriftserver2x”是ZooKeeper上的目录，表示客户端从该目录下随机选择JDBCServer实例进行连接。

    示例：安全模式下通过Beeline客户端连接时执行以下命令：

    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.<系统域名>@<系统域名>;"**
  - Keytab认证方式下的JDBCURL如下所示：
    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.*<系统域名>*@*<系统域名>*;user.principal=<principal_name>;user.keytab=<path_to_keytab>

    其中<principal_name>表示用户使用的Kerberos用户的principal，如“test@*<系统域名>*”。<path_to_keytab>表示<principal_name>对应的keytab文件路径，如“/opt/auth/test/user.keytab”。
- 普通模式下：
  jdbc:hive2://*<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>*/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=**sparkthriftserver2x**;

  示例：普通模式下通过Beeline客户端连接时执行以下命令：

  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;"**

非多主实例模式

非多主实例模式的客户端连接的是某个指定JDBCServer节点。该模式的连接字符串相比多主实例模式的去掉关于Zookeeper的参数项“serviceDiscoveryMode”和“zooKeeperNamespace”。

示例：安全模式下通过Beeline客户端连接非多主实例模式时执行以下命令：

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://***<server_IP>:<server_Port>***/;user.principal=spark2x/hadoop.<系统域名>@<系统域名>;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<系统域名>@<系统域名>;"**

说明

- 其中“<server_IP>:<server_Port>”是指定JDBCServer节点的URL。
- “CLIENT_HOME”是指客户端路径。

多主实例模式与非多主实例模式两种模式的JDBCServer接口相比，除连接方式不同外其他使用方法相同。由于Spark JDBCServer是Hive中的HiveServer2的另外一个实现，其使用方法，请参见Hive官网：**https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**。

6.31.3 Spark2x 多租户

背景介绍

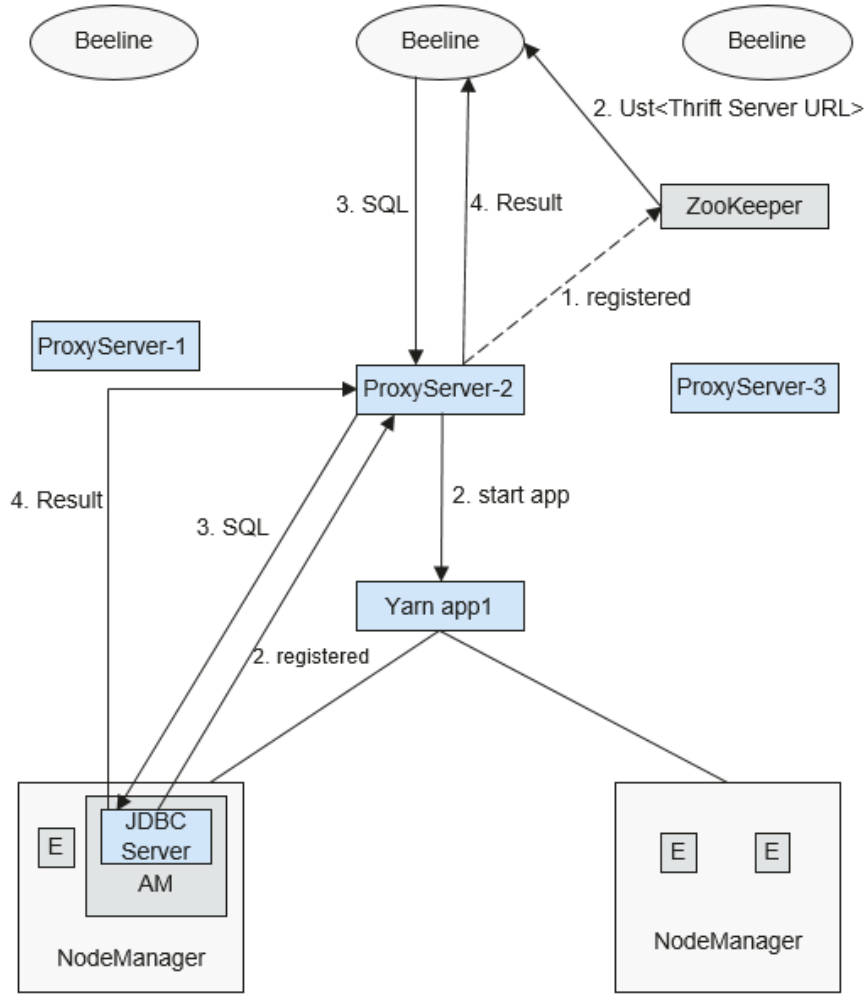
JDBCServer多主实例方案中，JDBCServer的实现使用YARN-Client模式，但YARN资源队列只有一个，为了解决这种资源局限的问题，引入了多租户模式。

多租户模式是将JDBCServer和租户绑定，每一个租户对应一个或多个JDBCServer，而一个JDBCServer只给一个租户提供服务。不同的租户可以配置不同的YARN队列，从而达到资源隔离，且JDBCServer根据需求动态启动，可避免浪费资源。

实现方案

多租户模式的HA方案原理如**图6-121**所示。

图 6-121 Spark JDBCServer 多租户



1. ProxyServer在启动时，向ZooKeeper注册自身消息，在指定目录中写入节点信息，节点信息包含了该实例对应的IP，端口，版本号和序列号等信息（多节点信息之间以逗号隔开）。

   说明

   多租户模式下，MRS页面上的JDBCServer实例是指ProxyServer（JDBCServer代理）。

   示例如下：

   ```
   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. 客户端连接ProxyServer时，需要指定Namespace，即访问ZooKeeper哪个目录下的ProxyServer实例。在连接的时候，会根据当前租户名的Hash值与Zookeeper下的Namespace实例个数取模获取连接的实例，详细URL参见**URL连接介绍**。
3. 客户端成功连接ProxyServer服务，ProxyServer服务首先确认是否有该租户的JDBCServer存在，如果有，直接将Beeline连上真正的JDBCServer；如果没有，则以YARN-Cluster模式启动一个新的JDBCServer。JDBCServer启动成功后，ProxyServer会获取JDBCServer的地址，并将Beeline连上JDBCServer。

4. 客户端发送SQL语句给ProxyServer，ProxyServer将语句转交给真正连上的JDBCServer处理。最后JDBCServer服务将结果返回给ProxyServer，ProxyServer再将结果返回给客户端。

在HA方案中，每个ProxyServer服务（即实例）都是独立且等同的，当其中一个实例在升级或者业务中断时，其他的实例也能接受客户端的连接请求。

URL 连接介绍

多租户模式

多租户模式的客户端读取ZooKeeper节点中的内容，连接对应的ProxyServer服务。连接字符串为：

- 安全模式下：
  - Kinit认证方式下的客户端URL如下所示：
    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.*<系统域名>*@*<系统域名>*;

    说明

    - 其中“<zkNode_IP>:<zkNode_Port>”是ZooKeeper的URL，多个URL以逗号隔开。
      例如：“192.168.81.37:2181,192.168.195.232:2181,192.168.169.84:2181”。
    - 其中sparkthriftserver2x是ZooKeeper上的目录，表示客户端从该目录下随机选择JDBCServer实例进行连接。

    示例：安全模式下通过Beeline客户端连接时执行以下命令：

    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.<系统域名>@<系统域名>;"**
  - Keytab认证方式下的URL如下所示：
    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.*<系统域名>*@*<系统域名>*;user.principal=<principal_name>;user.keytab=<path_to_keytab>

    其中<principal_name>表示用户使用的Kerberos用户的principal，如“test@*<系统域名>*”。<path_to_keytab>表示<principal_name>对应的keytab文件路径，如“/opt/auth/test/user.keytab”。
- 普通模式下：
  jdbc:hive2://*<zkNode1_IP>:<zkNode1_Port>,<zkNode2_IP>:<zkNode2_Port>,<zkNode3_IP>:<zkNode3_Port>*/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=**sparkthriftserver2x**;

  示例：普通模式下通过Beeline客户端连接时执行以下命令：

  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;"**

非多租户模式

非多租户模式的客户端连接的是某个指定JDBCServer节点。该模式的连接字符串相比多主实例模式的去掉关于ZooKeeper的参数项“serviceDiscoveryMode”和“zooKeeperNamespace”。

示例：安全模式下通过Beeline客户端连接非多租户模式时执行以下命令：

sh *CLIENT_HOME*/**spark/bin/beeline -u "jdbc:hive2://**<*server_IP*>:<*server_Port*>**/;user.principal=spark2x/hadoop.<系统域名>@<系统域名>;saslQop=auth-conf;auth=KERBEROS;principal=spark2x/hadoop.<系统域名>@<系统域名>;"**

说明

- 其中“<server_IP>:<server_Port>”是指定JDBCServer节点的URL。
- “CLIENT_HOME”是指客户端路径。

多租户模式与非多租户模式两种模式的JDBCServer接口相比，除连接方式不同外其他使用方法相同。由于Spark JDBCServer是Hive中的HiveServer2的另外一个实现，其使用方法，请参见Hive官网：**https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**。

指定租户

一般情况下，某用户提交的客户端会连接到该用户默认所属租户的JDBCServer上，若需要连接客户端到指定租户的JDBCServer上，可以通过添加**--hiveconf mapreduce.job.queuename**进行指定。

通过Beeline连接的命令示例如下（aaa为租户名称）：

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.<系统域名>@<系统域名>;'

6.31.4 Spark2x 与其他组件的关系

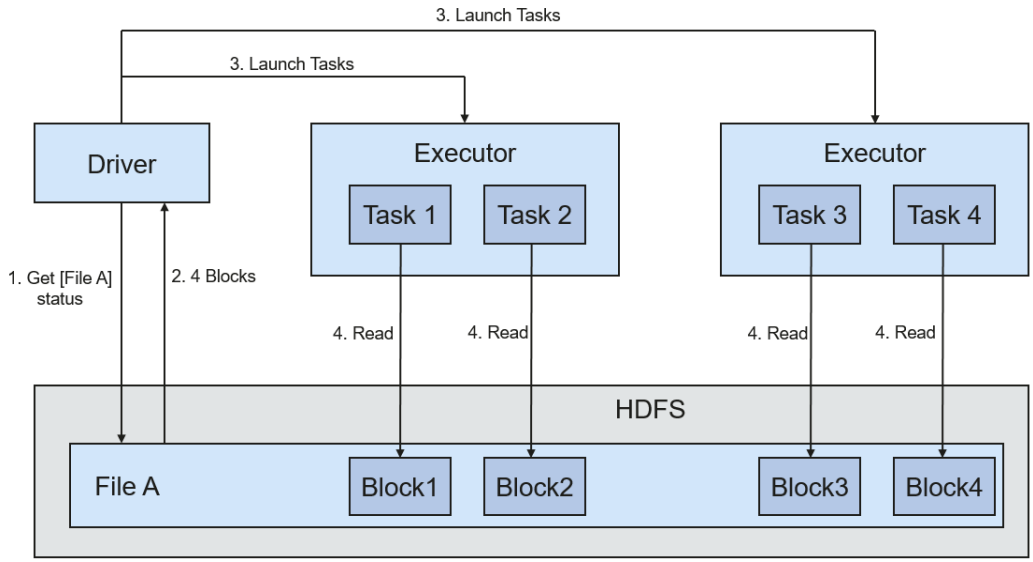
Spark 和 HDFS 的关系

通常，Spark中计算的数据可以来自多个数据源，如Local File、HDFS等。最常用的是HDFS，用户可以一次读取大规模的数据进行并行计算。在计算完成后，也可以将数据存储到HDFS。

分解来看，Spark分成控制端(Driver)和执行端（Executor）。控制端负责任务调度，执行端负责任务执行。

读取文件的过程如图6-122所示。

图 6-122 读取文件过程

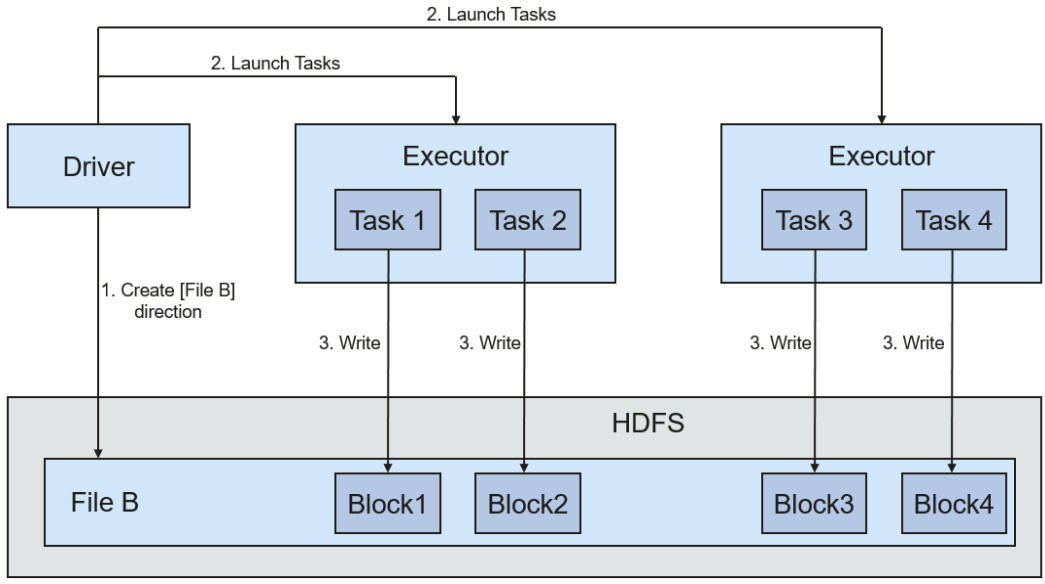


读取文件步骤的详细描述如下所示：

1. Driver与HDFS交互获取File A的文件信息。
2. HDFS返回该文件具体的Block信息。
3. Driver根据具体的Block数据量，决定一个并行度，创建多个Task去读取这些文件Block。
4. 在Executor端执行Task并读取具体的Block，作为RDD（弹性分布数据集）的一部分。

写入文件的过程如图6-123所示。

图 6-123 写入文件过程



HDFS文件写入的详细步骤如下所示：

1. Driver创建要写入文件的目录。
2. 根据RDD分区分块情况，计算出写数据的Task数，并下发这些任务到Executor。
3. Executor执行这些Task，将具体RDD的数据写入到步骤1创建的目录下。

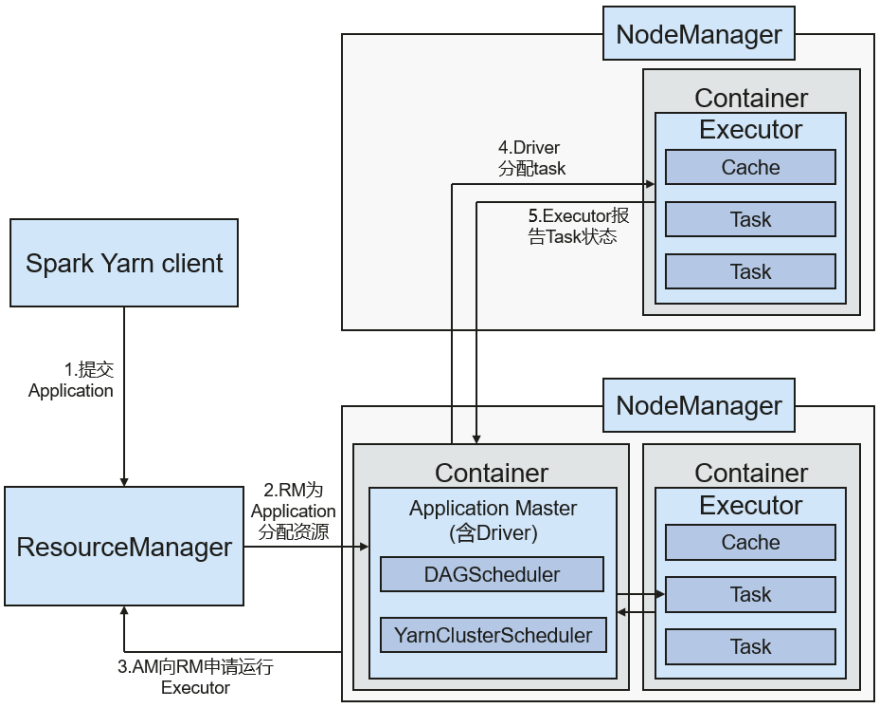
Spark 和 YARN 的关系

Spark的计算调度方式，可以通过YARN的模式实现。Spark共享YARN集群提供丰富的计算资源，将任务分布式的运行起来。Spark on YARN分两种模式：YARN Cluster和YARN Client。

- YARN Cluster模式

  运行框架如图6-124所示。

  图 6-124 Spark on yarn-cluster 运行框架



  Spark on YARN-Cluster实现流程：

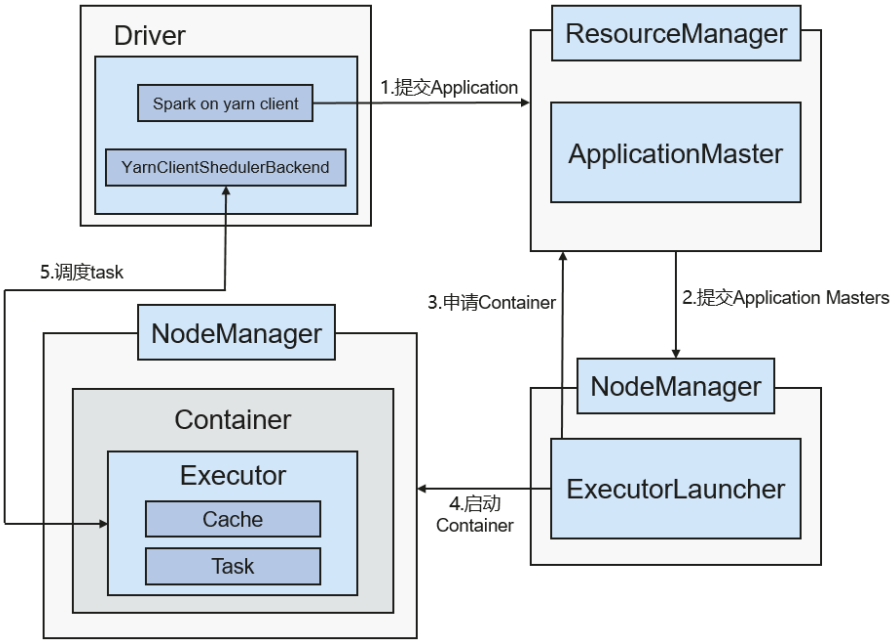
  a. 首先由客户端生成Application信息，提交给ResourceManager。
  b. ResourceManager为Spark Application分配第一个Container(ApplicationMaster)，并在该Container上启动Driver。
  c. ApplicationMaster向ResourceManager申请资源以运行Container。

     ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。
  d. Driver分配Task给Executor执行。
  e. Executor执行Task并向Driver汇报运行状况。

- YARN Client模式

  运行框架如图6-125所示。

图 6-125 Spark on yarn-client 运行框架



Spark on YARN-Client实现流程：

说明

在YARN-Client模式下，Driver部署在Client端，在Client端启动。YARN-Client模式下，不兼容老版本的客户端。推荐使用YARN-Cluster模式。

a. 客户端向ResourceManager发送Spark应用提交请求，Client端将启动ApplicationMaster所需的所有信息打包，提交给ResourceManager上，ResourceManager为其返回应答，该应答中包含多种信息（如ApplicationId、可用资源使用上限和下限等）。ResourceManager收到请求后，会为ApplicationMaster寻找合适的节点，并在该节点上启动它。ApplicationMaster是Yarn中的角色，在Spark中进程名字是ExecutorLauncher。

b. 根据每个任务的资源需求，ApplicationMaster可向ResourceManager申请一系列用于运行任务的Container。

c. 当ApplicationMaster（从ResourceManager端）收到新分配的Container列表后，会向对应的NodeManager发送信息以启动Container。

   ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。

   说明

   正在运行的Container不会被挂起释放资源。

d. Driver分配Task给Executor执行。Executor执行Task并向Driver汇报运行状况。

6.31.5 Spark2x 开源新特性说明

Spark2x版本相对于Spark 1.5版本新增了一些开源特性。

具体特性或相关概念如下：

- DataSet，详见**SparkSQL和DataSet原理**。
- Spark SQL Native DDL/DML，详见**SparkSQL和DataSet原理**。
- SparkSession，详见**SparkSession原理**。
- Structured Streaming，详见**Structured Streaming原理**。
- 小文件优化。
- 聚合算法优化。
- Datasource表优化。
- 合并CBO优化。

6.31.6 Spark 跨源复杂数据的 SQL 查询优化

场景描述

出于管理和信息收集的需要，企业内部会存储海量数据，包括数目众多的各种数据库、数据仓库等，此时会面临以下困境：数据源种类繁多，数据集结构化混合，相关数据存放分散等，这就导致了跨源复杂查询因传输效率低，耗时长。

当前开源Spark在跨源查询时，只能对简单的filter进行下推，因此造成大量不必要的数据传输，影响SQL引擎性能。针对下推能力进行增强，当前对aggregate、复杂projection、复杂predicate均可以下推到数据源，尽量减少不必要数据的传输，提升查询性能。

目前仅支持JDBC数据源的查询下推，支持的下推模块有aggregate、projection、predicate、aggregate over inner join、aggregate over union all等。为应对不同应用场景的特殊需求，对所有下推模块设计开关功能，用户可以自行配置是否应用上述查询下推的增强。

表 6-26 跨源查询增加特性对比

模块	增强前	增强后
aggregate	不支持aggregate下推	• 支持的聚合函数为：sum, avg, max, min, count 例如：select count(*) from table • 支持聚合函数内部表达式 例如：select sum(a+b) from table • 支持聚合函数运算，例如：select avg(a) + max(b) from table • 支持having下推 例如：select sum(a) from table where a>0 group by b having sum(a)>10 • 支持部分函数下推 支持对abs()、month()、length()等数学、时间、字符串函数进行下推。并且，除了以上内置函数，用户还可以通过SET命令新增数据源支持的函数。 例如：select sum(abs(a)) from table • 支持aggregate之后的limit、order by下推（由于Oracle不支持limit，所以Oracle中limit、order by不会下推） 例如：select sum(a) from table where a>0 group by b order by sum(a) limit 5
projection	仅支持简单projection下推，例如：select a, b from table	• 支持复杂表达式下推。 例如：select (a+b)*c from table • 支持部分函数下推，详细参见表下方的说明。 例如：select length(a)+abs(b) from table • 支持projection之后的limit、order by下推。 例如：select a, b+c from table order by a limit 3
predicate	仅支持运算符左边为列名右边为值的简单filter，例如 select * from table where a>0 or b in（“aaa”，“bbb”）	• 支持复杂表达数下推 例如：select * from table where a+b>c*d or a/c in (1, 2, 3) • 支持部分函数下推，详细参见表下方的说明。 例如：select * from table where length(a)>5

模块	增强前	增强后
aggregate over inner join	需要将两个表中相关的数据全部加载到Spark，先进行join操作，再进行aggregate操作	支持以下几种： • 支持的聚合函数为：sum, avg, max, min,count • 所有aggregate只能来自同一个表，group by可以来自一个表或者两个表，只支持inner join。 不支持的情形有： • 不支持aggregate同时来自join左表和右表的下推。 • 不支持aggregate内包含运算，如：sum(a+b)。 • 不支持aggregate运算，如：sum(a)+min(b)。
aggregate over union all	需要将两个表中相关的数据全部加载到Spark，先进行union操作，再进行aggregate操作	支持情况： 支持的聚合函数为：sum, avg, max, min,count 不支持的情况： • 不支持aggregate内包含运算，如：sum(a+b)。 • 不支持aggregate运算，如：sum(a)+min(b)。

注意事项

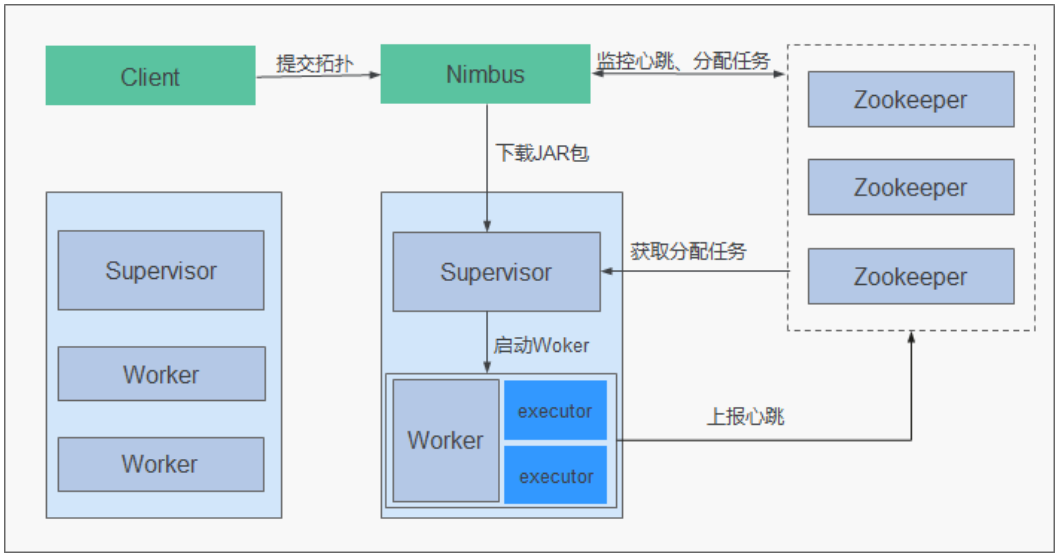
- 外部数据源是Hive的场景，通过Spark建的外表无法进行查询。
- 数据源只支持MySQL和Mppdb。

6.32 Storm

6.32.1 Storm 基本原理

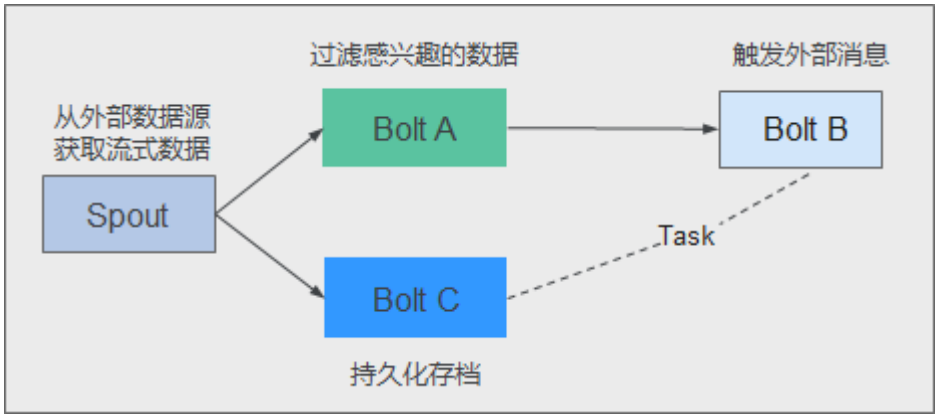
Apache Storm是一个分布式、可靠、容错的实时流式数据处理的系统。在Storm中，先要设计一个用于实时计算的图状结构，称之为拓扑（topology）。这个拓扑将会被提交给集群，由集群中的主控节点（master node）分发代码，将任务分配给工作节点（worker node）执行。一个拓扑中包括spout和bolt两种角色，其中spout发送消息，负责将数据流以tuple元组的形式发送出去；而bolt则负责转换这些数据流，在bolt中可以完成计算、过滤等操作，bolt自身也可以随机将数据发送给其他bolt。由spout发射出的tuple是不可变数组，对应着固定的键值对。

图 6-126 Storm 基本架构



业务处理逻辑被封装进Storm中的Topology中。一个Topology是由一组Spout组件（数据源）和Bolt组件（逻辑处理）通过Stream Groupings进行连接的有向无环图（DAG）。Topology里面的每一个Component（Spout/Bolt）节点都是并行运行的。在Topology里面， 可以指定每个节点的并行度，Storm则会在集群里面分配相应的Task来同时计算，以增强系统的处理能力。

图 6-127 Topology



Storm有众多适用场景：实时分析、持续计算、分布式ETL等。Storm有如下几个特点：

- 适用场景广泛
- 易扩展，可伸缩性高
- 保证无数据丢失
- 容错性好
- 易于构建和操控
- 多语言

Storm作为计算平台，在业务层为用户提供了更为易用的业务实现方式：CQL（Continuous Query Language—持续查询语言）。CQL具有以下几个特点：

- 使用简单：CQL语法和标准SQL语法类似，只要具备SQL基础，通过简单地学习，即可快速地进行业务开发。
- 功能丰富：CQL除了包含标准SQL的各类基本表达式等功能之外，还特别针对流处理场景增加了窗口、过滤、并发度设置等功能。
- 易于扩展：CQL提供了拓展接口，以支持日益复杂的业务场景，用户可以自定义输入、输出、序列化、反序列化等功能来满足特定的业务场景
- 易于调试：CQL提供了详细的异常码说明，降低了用户对各种错误的处理难度。

关于Storm的架构和详细原理介绍，请参见：**https://storm.apache.org/**。

更多关于Storm组件操作指导，请参考**使用Storm**。

Storm 原理

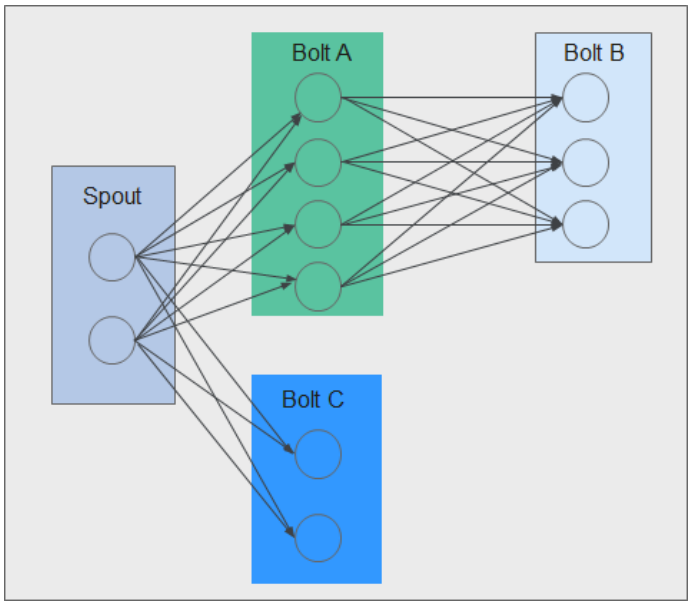
- **基本概念**

表 6-27 概念介绍

概念	说明
Tuple	Storm核心数据结构，是消息传递的基本单元，不可变Key-Value对，这些Tuple会以一种分布式的方式进行创建和处理。
Stream	Storm的关键抽象，是一个无边界的连续Tuple序列。
Topology	在Storm平台上运行的一个实时应用程序，由各个组件（Component）组成的一个DAG（Directed Acyclic Graph）。一个Topology可以并发地运行在多台机器上，每台机器上可以运行该DAG中的一部分。Topology与Hadoop中的MapReduce Job类似，不同的是，它是一个长驻程序，一旦开始就不会停止，除非人工中止。
Spout	Topology中产生源数据的组件，是Tuple的来源，通常可以从外部数据源（如消息队列、数据库、文件系统、TCP连接等）读取数据，然后转换为Topology内部的数据结构Tuple，由下一级组件处理。
Bolt	Topology中接受数据并执行具体处理逻辑（如过滤，统计、转换、合并、结果持久化等）的组件。
Worker	是Topology运行态的物理进程。每个Worker是一个JVM进程，每个Topology可以由多个Worker并行执行，每个Worker运行Topology中的一个逻辑子集。
Task	Worker中每一个Spout/Bolt的线程称为一个Task。
Stream groupings	Storm中的Tuple分发策略，即后一级Bolt以什么分发方式来接收数据。当前支持的策略有：Shuffle Grouping， Fields Grouping， All Grouping， Global Grouping， Non Grouping， Directed Grouping。

图6-128描述了一个由Spout、Bolt组成的DAG，即Topology。图中每个矩形框代表Spout或者Bolt，矩形框内的节点表示各个并发的Task，Task之间的“边”代表数据流——Stream。

图 6-128 Topology 示意图



- **可靠性**

  Storm提供三种级别的数据可靠性：

  - 至多一次：处理的数据可能会丢失，但不会被重复处理。此情况下，系统吞吐量最大。
  - 至少一次：保证数据传输可靠，但可能会被重复处理。此情况下，对在超时时间内没有获得成功处理响应的数据，会在Spout处进行重发，供后续Bolt再次处理，会对性能稍有影响。
  - 精确一次：数据成功传递，不丢失，不冗余处理。此情况下，性能最差。

  可靠性不同级别的选择，需要根据业务对可靠性的要求来选择、设计。例如对于一些对数据丢失不敏感的业务，可以在业务中不考虑数据丢失处理从而提高系统性能；而对于一些严格要求数据可靠性的业务，则需要使用精确一次的可靠性方案，以确保数据被处理且仅被处理一次。

- **容错**

  Storm是一个容错系统，提供较高可用性。表6-28从Storm的不同部件失效的情况角度解释其容错能力：

表 6-28 容错能力

失效场景	说明
Nimbus失效	Nimbus是无状态且快速失效的。当主Nimbus失效时，备Nimbus会接管，并对外提供服务。
Supervisor失效	Supervisor是工作节点的后台守护进程，是一种快速失效机制，且是无状态的，并不影响正在该节点上运行的Worker，但是会无法接收新的Worker分配。当Supervisor失效时，OMS会侦测到，并及时重启该进程。

失效场景	说明
Worker失效	该Worker所在节点上的Supervisor会在此节点上重新启动该Worker。如果多次重启失败，则Nimbus会将该任务重新分配到其他节点。
节点失效	该节点上的所有分配的任务会超时，而Nimbus会将这些Worker重新分配到其他节点。

Storm 开源特性

- 分布式实时计算框架

  开源Storm集群中的每台机器上都可以运行多个工作进程，每个工作进程又可创建多个线程，每个线程可以执行多个任务，任务是并发进行数据处理。
- 高容错

  如果在消息处理过程中有节点、进程等出现异常，提供重新部署该处理单元的能力。
- 可靠的消息保证

  支持At-Least Once、At-Most Once、Exactly Once的数据处理模式。
- 安全机制

  提供基于Kerberos的认证以及可插拔的授权机制，提供支持SSL的Storm UI以及Log Viewer界面，同时支持与大数据平台其他组件（如ZooKeeper，HDFS等）进行安全集成。
- 灵活的拓扑定义及部署

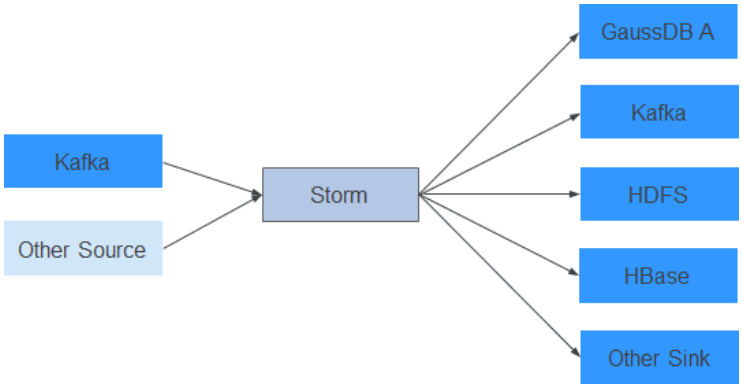
  使用Flux框架定义及部署业务拓扑，在业务DAG发生变化时，只需对YAML DSL（domain-specific language）定义进行修改，无需重新编译及打包业务代码。
- 与外部组件集成

  支持与多种外部组件集成，包括：Kafka、HDFS、HBase、Redis或JDBC/RDBMS等服务，便于实现涉及多种数据源的业务。

6.32.2 Storm 与其他组件的关系

Storm，提供实时的分布式计算框架，它可以从数据源（如Kafka、TCP连接等）中获得实时消息数据，在实时平台上完成高吞吐、低延迟的实时计算，并将结果输出到消息队列或者进行持久化。Storm与其他组件的关系如图6-129所示：

图 6-129 组件关系图



Storm 和 Streaming 的关系

Storm和Streaming都使用的开源Apache Storm内核，不同的是，Storm使用的内核版本是1.2.1，Streaming使用的是0.10.0。Streaming组件一般用来在升级场景继承过渡业务，比如之前版本已经部署Streaming并且有业务在运行的情况下，升级后仍然可以使用Streaming。如果是新搭建的集群，则建议使用Storm。

Storm 1.2.1新增特性说明：

- **分布式缓存**：提供命令行工具共享和更新拓扑的所需要的外部资源（配置），无需重新打包和部署拓扑。
- **Native Streaming Window API**：提供基于窗口的API。
- **资源调度器**：新增基于资源的调度器插件，可以在拓扑定义时指定可使用的最大资源，并且通过配置的方式指定用户的资源配额，从而管理该用户名下的拓扑资源。
- **State Management**：提供带检查点机制的Bolt接口，当事件失败时，Storm会自动管理bolt的状态并且执行恢复。
- **消息采样和调试：**在Storm UI界面可以开关拓扑或者组件级别的调试，将流消息按采样比率输出到指定日志中。
- **Worker动态分析：**在Storm UI界面可以收集Worker进程的Jstack、Heap日志，并且可以重启Worker进程。
- **拓扑日志级别动态调整：**提供命令行和Storm UI两种方式对运行中的拓扑日志进行动态修改。
- **性能提升：**与之前的版本相比，Storm的性能得到了显著提升。虽然，拓扑的性能和用例场景及外部服务的依赖有很大的关系，但是对于大多数场景来说，性能可以提升3倍。

6.32.3 Storm 开源增强特性

- CQL

  CQL（Continuous Query Language），持续查询语言，是一种用于实时数据流上的查询语言，它是一种SQL-like的语言，相对于SQL，CQL中增加了（时序）窗口的概念，将待处理的数据保存在内存中，进行快速的内存计算，CQL的输出结果为数据流在某一时刻的计算结果。使用CQL，可以快速进行业务开发，并方便地将业务提交到Storm平台开启实时数据的接收、处理及结果输出；并可以在合适的时候中止业务。

- 高可用性

  Nimbus HA机制，避免了开源Storm集群中Nimbus出现单点故障而导致集群无法提供Topology的新增及管理操作的问题，增强了集群可用性。

6.33 Tez

Tez是Apache最新的支持DAG（有向无环图）作业的开源计算框架，它可以将多个有依赖的作业转换为一个作业从而大幅提升DAG作业的性能。

MRS将Tez作为Hive的默认执行引擎，执行效率远远超过原先的MapReduce的计算引擎。

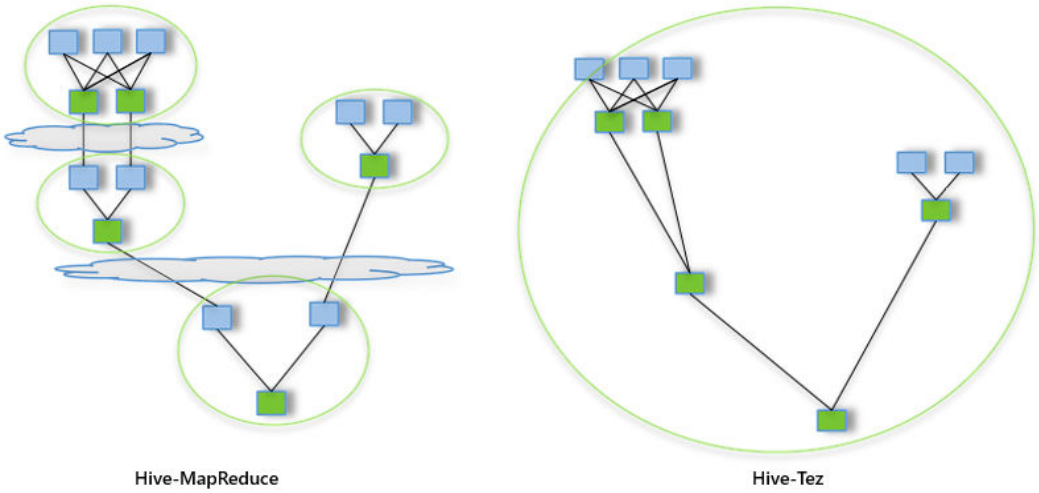
有关Tez的详细说明，请参见：**https://tez.apache.org/**。

更多关于Tez组件操作指导，请参考**使用Tez**。

Tez 和 MapReduce 间的关系

Tez采用了DAG来组织MapReduce任务（DAG中一个节点就是一个RDD，边表示对RDD的操作）。它的核心思想是把将Map任务和Reduce任务进一步拆分，Map任务拆分为Input-Processor-Sort-Merge-Output，Reduce任务拆分为Input-Shuffle-Sort-Merge-Process-output，Tez将若干小任务灵活重组，形成一个大的DAG作业。

图 6-130 Hive 基于 MapReduce 提交任务和基于 Tez 提交任务流程图



Hive on MapReduce任务中包含多个MapReduce任务，每个任务都会将中间结果存储到HDFS上——前一个步骤中的reducer为下一个步骤中的mapper提供数据。Hive on Tez任务仅在一个任务中就能完成同样的处理过程，任务之间不需要访问HDFS。

Tez 和 Yarn 间的关系

Tez是运行在Yarn之上的计算框架，运行时环境由Yarn的ResourceManager和ApplicationMaster组成。其中ResourceManager是一个全新的资源管理系统，而ApplicationMaster则负责MapReduce作业的数据切分、任务划分、资源申请和任务调度与容错等工作。此外，TezUI依赖Yarn提供的TimelineServer实现Tez任务运行过程呈现。

6.34 YARN

6.34.1 YARN 基本原理

为了实现一个Hadoop集群的集群共享、可伸缩性和可靠性，并消除早期MapReduce框架中的JobTracker性能瓶颈，开源社区引入了统一的资源管理框架YARN。

YARN是将JobTracker的两个主要功能（资源管理和作业调度/监控）分离，主要方法是创建一个全局的ResourceManager（RM）和若干个针对应用程序的ApplicationMaster（AM）。

更多关于Yarn组件操作指导，请参考**使用Yarn**。

说明

- 如需使用YARN，请确保MRS集群内已安装Hadoop服务。
- 应用程序是指传统的MapReduce作业或作业的DAG（有向无环图）。

YARN 结构

YARN分层结构的本质是ResourceManager。这个实体控制整个集群并管理应用程序向基础计算资源的分配。ResourceManager将各个资源部分（计算、内存、带宽等）精心安排给基础NodeManager（YARN的每个节点代理）。ResourceManager还与Application Master一起分配资源，与NodeManager一起启动和监视它们的基础应用程序。在此上下文中，Application Master承担了以前的TaskTracker的一些角色，ResourceManager承担了JobTracker的角色。

Application Master管理一个在YARN内运行的应用程序的每个实例。Application Master负责协调来自ResourceManager的资源，并通过NodeManager监视容器的执行和资源使用（CPU、内存等的资源分配）。

NodeManager管理一个YARN集群中的每个节点。NodeManager提供针对集群中每个节点的服务，从监督对一个容器的终生管理到监视资源和跟踪节点健康。MRv1通过插槽管理Map和Reduce任务的执行，而NodeManager管理抽象容器，这些容器代表着可供一个特定应用程序使用的针对每个节点的资源。

图 6-131 YARN 结构

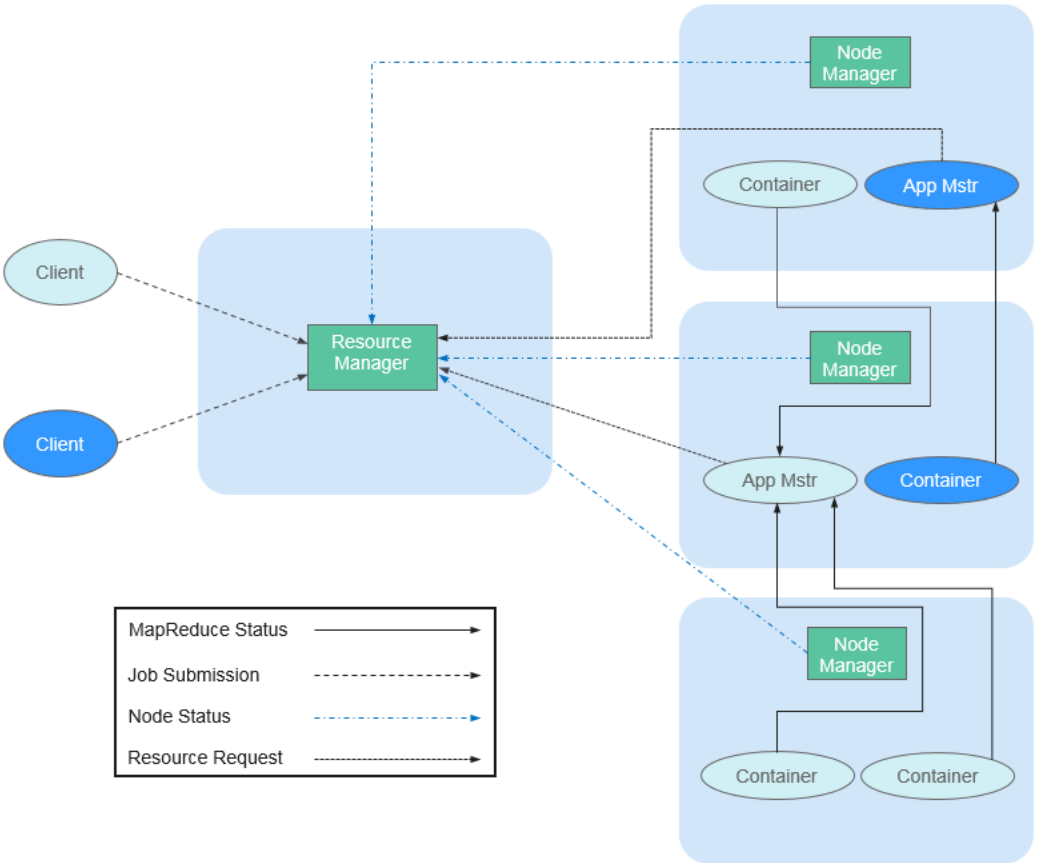


图6-131中各部分的功能如表6-29所示。

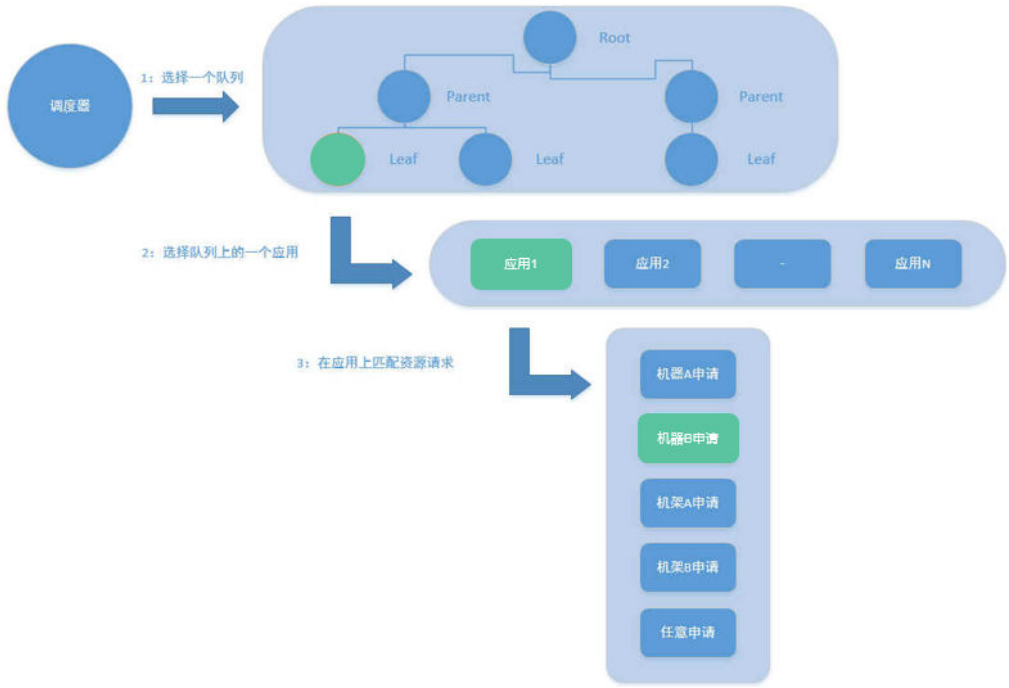
表 6-29 结构图说明

名称	描述
Client	YARN Application客户端，用户可以通过客户端向ResourceManager提交任务，查询Application运行状态等。
ResourceManager(RM)	负责集群中所有资源的统一管理和分配。接收来自各个节点（NodeManager）的资源汇报信息，并根据收集的资源按照一定的策略分配给各个应用程序。
NodeManager(NM)	NodeManager（NM）是YARN中每个节点上的代理，管理Hadoop集群中单个计算节点，包括与ResourceManager保持通信，监督Container的生命周期管理，监控每个Container的资源使用（内存、CPU等）情况，追踪节点健康状况，管理日志和不同应用程序用到的附属服务（auxiliary service）。
ApplicationMaster(AM)	即图中的App Mstr，负责一个Application生命周期内的所有工作。包括：与RM调度器协商以获取资源；将得到的资源进一步分配给内部任务（资源的二次分配）；与NM通信以启动/停止任务；监控所有任务运行状态，并在任务运行失败时重新为任务申请资源以重启任务。
Container	Container是YARN中的资源抽象，封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等（目前仅封装内存和CPU），当AM向RM申请资源时，RM为AM返回的资源便是用Container表示。YARN会为每个任务分配一个Container，且该任务只能使用该Container中描述的资源。

在YARN中，资源调度器是以层级队列方式组织资源的，这种组织方式有利于资源在不同队列间分配和共享，进而提高集群资源利用率。如下图所示，Superior Scheduler和Capacity Scheduler的核心资源分配模型相同。

调度器会维护队列的信息。用户可以向一个或者多个队列提交应用。每次NM心跳的时候，调度器会根据一定规则选择一个队列，再选择队列上的一个应用，并尝试在这个应用上分配资源。若因参数限制导致分配失败，将选择下一个应用。选择一个应用后，调度器会处理此应用的资源申请。其优先级从高到低依次为：本地资源的申请、同机架的申请，任意机器的申请。

图 6-132 资源分配模型



YARN 原理

新的Hadoop MapReduce框架被命名为MRv2或YARN。YARN主要包括ResourceManager、ApplicationMaster与NodeManager三个部分。

- ResourceManager：RM是一个全局的资源管理器，负责整个系统的资源管理和分配。主要由两个组件构成：调度器（Scheduler）和应用程序管理器（Applications Manager）。
  - 调度器根据容量、队列等限制条件（如每个队列分配一定的资源，最多执行一定数量的作业等），将系统中的资源分配给各个正在运行的应用程序。调度器仅根据各个应用程序的资源需求进行资源分配，而资源分配单位用一个抽象概念Container表示。Container是一个动态资源分配单位，将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。此外，该调度器是一个可插拔的组件，用户可根据自己的需要设计新的调度器，YARN提供了多种直接可用的调度器，比如Fair Scheduler和Capacity Scheduler等。
  - 应用程序管理器负责管理整个系统中所有应用程序，包括应用程序提交、与调度器协商资源以启动ApplicationMaster、监控ApplicationMaster运行状态并在失败时重新启动等。
- NodeManager：NM是每个节点上的资源和任务管理器，一方面，会定时向RM汇报本节点上的资源使用情况和各个Container的运行状态；另一方面，接收并处理来自AM的Container启动/停止等请求。
- ApplicationMaster：AM负责一个Application生命周期内的所有工作。包括：
  - 与RM调度器协商以获取资源。
  - 将得到的资源进一步分配给内部的任务（资源的二次分配）。
  - 与NM通信以启动/停止任务。
  - 监控所有任务运行状态，并在任务运行失败时重新为任务申请资源以重启任务。

开源容量调度器 Capacity Scheduler 原理

Capacity Scheduler是一种多用户调度器，它以队列为单位划分资源，为每个队列设定了资源最低保证和使用上限。同时，也为每个用户设定了资源使用上限以防止资源滥用。而当一个队列的资源有剩余时，可暂时将剩余资源共享给其他队列。

Capacity Scheduler支持多个队列，为每个队列配置一定的资源量，并采用FIFO调度策略。为防止同一用户的应用独占队列资源，Capacity Scheduler会对同一用户提交的作业所占资源量进行限定。调度时，首先计算每个队列使用的资源，选择使用资源最少的队列；然后按照作业优先级和提交时间顺序选择，同时考虑用户资源量的限制和内存限制。Capacity Scheduler主要有如下特性：

- 容量保证。MRS集群管理员可为每个队列设置资源最低保证和资源使用上限，而所有提交到队列的应用程序共享这些资源。
- 灵活性。如果一个队列中的资源有剩余，可以暂时共享给那些需要资源的队列，而一旦该队列有新的应用程序提交，则占用资源的队列将资源释放给该队列。这种资源灵活分配的方式可明显提高资源利用率。
- 多重租赁。支持多用户共享集群和多应用程序同时运行。为防止单个应用程序、用户或者队列独占集群中的资源，MRS集群管理员可为之增加多重约束（比如单个应用程序同时运行的任务数等）。
- 安全保证。每个队列有严格的ACL列表规定它的访问用户，每个用户可指定哪些用户允许查看自己应用程序的运行状态或者控制应用程序。此外，MRS集群管理员可指定队列管理员和集群系统管理员。
- 动态更新配置文件。MRS集群管理员可根据需要动态修改配置参数以实现在线集群管理。

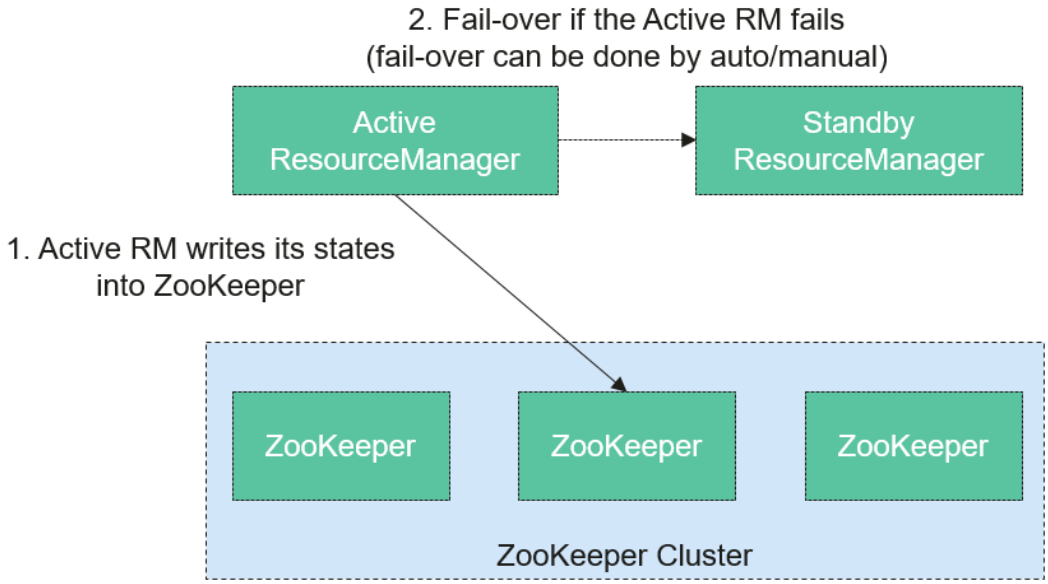
Capacity Scheduler中每个队列可以限制资源使用量。队列间的资源分配以使用量作为排列依据，使得容量小的队列有竞争优势。集群整体吞吐较大，延迟调度机制使得应用可以有机会放弃跨机器或者跨机架的调度，争取本地调度。

6.34.2 YARN HA 方案介绍

YARN HA 原理与实现方案

YARN中的ResourceManager负责整个集群的资源管理和任务调度，在Hadoop2.4版本之前，ResourceManager在YARN集群中存在单点故障的问题。YARN高可用性方案通过引入冗余的ResourceManager节点的方式，解决了这个基础服务的可靠性和容错性问题。

图 6-133 ResourceManager 高可用性实现架构



ResourceManager的高可用性方案是通过设置一组Active/Standby的ResourceManager节点来实现的（如图6-133）。与HDFS的高可用性方案类似，任何时间点上都只能有一个ResourceManager处于Active状态。当Active状态的ResourceManager发生故障时，可通过自动或手动的方式触发故障转移，进行Active/Standby状态切换。

在未开启自动故障转移时，YARN集群启动后，MRS集群管理员需要在命令行中使用***yarn rmadmin***命令手动将其中一个ResourceManager切换为Active状态。当需要执行计划性维护或故障发生时，则需要先手动将Active状态的ResourceManager切换为Standby状态，再将另一个ResourceManager切换为Active状态。

开启自动故障转移后，ResourceManager会通过内置的基于ZooKeeper实现的ActiveStandbyElector来决定哪一个ResourceManager应该成为Active节点。当Active状态的ResourceManager发生故障时，另一个ResourceManager将自动被选举为Active状态以接替故障节点。

当集群的ResourceManager以HA方式部署时，客户端使用的“yarn-site.xml”需要配置所有ResourceManager地址。客户端（包括ApplicationMaster和NodeManager）会以轮询的方式寻找Active状态的ResourceManager，也就是说客户端需要自己提供容错机制。如果当前Active状态的ResourceManager无法连接，那么会继续使用轮询的方式找到新的ResourceManager。

备RM升主后，能够恢复故障发生时上层应用运行的状态（详见**ResourceManager Restart**）。当启用ResourceManager Restart时，重启后的ResourceManager就可以通过加载之前Active的ResourceManager的状态信息，并通过接收所有NodeManager上container的状态信息重构运行状态继续执行。这样应用程序通过定期执行检查点操作保存当前状态信息，就可以避免工作内容的丢失。状态信息需要让Active/Standby的ResourceManager都能访问。当前系统提供了三种共享状态信息的方法：通过文件系统共享（FileSystemRMStateStore）、通过LevelDB数据库共享（LeveldbRMStateStore）或通过ZooKeeper共享（ZKRMStateStore）。这三种方式中只有ZooKeeper共享支持Fencing机制。Hadoop默认使用ZooKeeper共享。

关于YARN高可用性方案的更多信息，可参考如下链接：

MRS 3.2.0之前版本：**http://hadoop.apache.org/docs/r3.1.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html**

MRS 3.2.0及之后版本：https://hadoop.apache.org/docs/r3.3.1/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

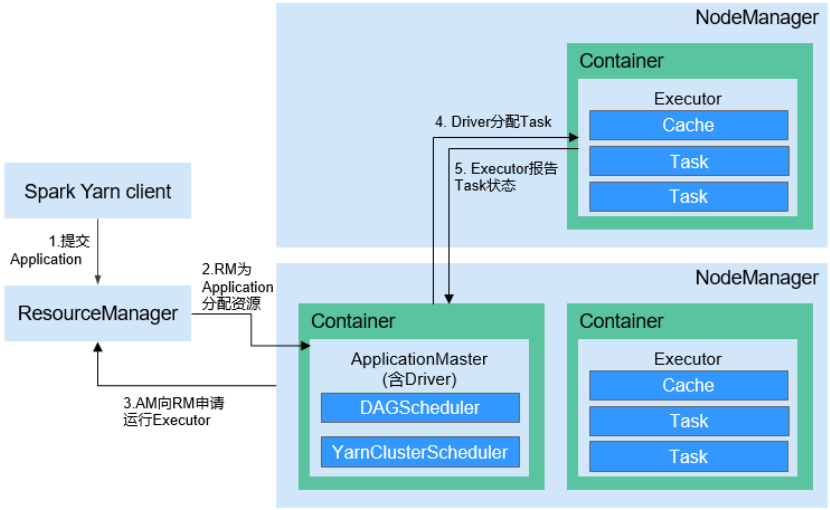
6.34.3 Yarn 与其他组件的关系

Yarn 和 Spark 组件的关系

Spark的计算调度方式，可以通过Yarn的模式实现。Spark共享Yarn集群提供丰富的计算资源，将任务分布式的运行起来。Spark on Yarn分两种模式：Yarn Cluster和Yarn Client。

- Yarn Cluster模式
  运行框架如图6-134所示。

  图 6-134 Spark on yarn-cluster 运行框架

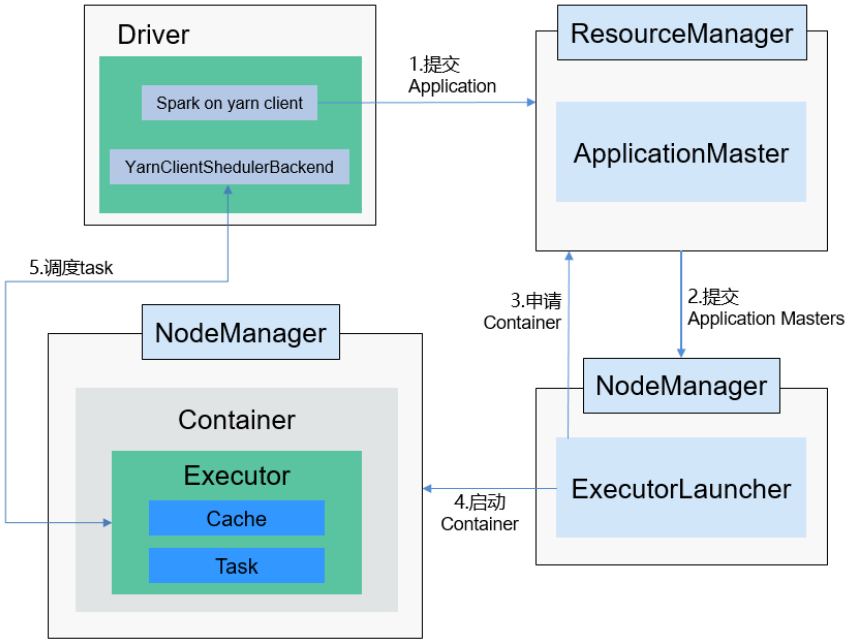


  Spark on yarn-cluster实现流程：

  a. 首先由客户端生成Application信息，提交给ResourceManager。
  b. ResourceManager为Spark Application分配第一个Container(ApplicationMaster)，并在该Container上启动Driver。
  c. ApplicationMaster向ResourceManager申请资源以运行Container。
     ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。
  d. Driver分配Task给Executor执行。
  e. Executor执行Task并向Driver汇报运行状况。

- Yarn Client模式
  运行框架如图6-135所示。

图 6-135 Spark on yarn-client 运行框架



Spark on yarn-client实现流程：

说明

在yarn-client模式下，Driver部署在Client端，在Client端启动。yarn-client模式下，不兼容老版本的客户端。推荐使用yarn-cluster模式。

a. 客户端向ResourceManager发送Spark应用提交请求，ResourceManager为其返回应答，该应答中包含多种信息（如ApplicationId、可用资源使用上限和下限等）。Client端将启动ApplicationMaster所需的所有信息打包，提交给ResourceManager上。

b. ResourceManager收到请求后，会为ApplicationMaster寻找合适的节点，并在该节点上启动它。ApplicationMaster是Yarn中的角色，在Spark中进程名字是ExecutorLauncher。

c. 根据每个任务的资源需求，ApplicationMaster可向ResourceManager申请一系列用于运行任务的Container。

d. 当ApplicationMaster（从ResourceManager端）收到新分配的Container列表后，会向对应的NodeManager发送信息以启动Container。

   ResourceManager分配Container给ApplicationMaster，ApplicationMaster和相关的NodeManager通讯，在获得的Container上启动Executor，Executor启动后，开始向Driver注册并申请Task。

   说明

   正在运行的Container不会被挂起释放资源。

e. Driver分配Task给Executor执行。Executor执行Task并向Driver汇报运行状况。

Yarn 和 MapReduce 的关系

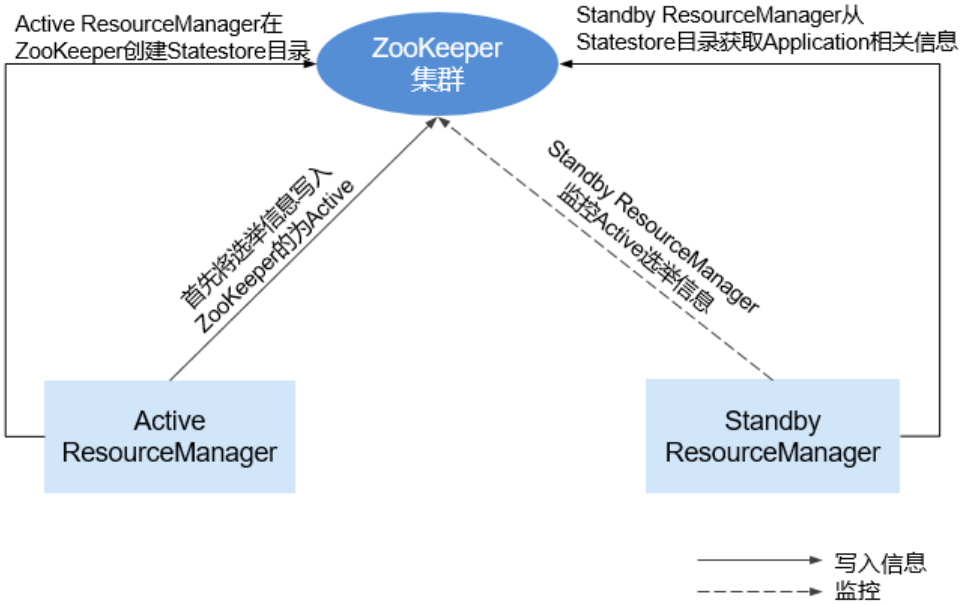
MapReduce是运行在Yarn之上的一个批处理的计算框架。MRv1是Hadoop 1.0中的MapReduce实现，它由编程模型（新旧编程接口）、运行时环境（由JobTracker和

TaskTracker组成）和数据处理引擎（MapTask和ReduceTask）三部分组成。该框架在扩展性、容错性（JobTracker单点）和多框架支持（仅支持MapReduce一种计算框架）等方面存在不足。MRv2是Hadoop 2.0中的MapReduce实现，它在源码级重用了MRv1的编程模型和数据处理引擎实现，但运行时环境由Yarn的ResourceManager和ApplicationMaster组成。其中ResourceManager是一个全新的资源管理系统，而ApplicationMaster则负责MapReduce作业的数据切分、任务划分、资源申请和任务调度与容错等工作。

Yarn 和 ZooKeeper 的关系

ZooKeeper与Yarn的关系如图6-136所示。

图 6-136 ZooKeeper 与 Yarn 的关系



1. 在系统启动时，ResourceManager会尝试把选举信息写入ZooKeeper，第一个成功写入ZooKeeper的ResourceManager被选举为Active ResourceManager，另一个为Standby ResourceManager。Standby ResourceManager定时去ZooKeeper监控Active ResourceManager选举信息。
2. Active ResourceManager还会在ZooKeeper中创建Statestore目录，存储Application相关信息。当Active ResourceManager产生故障时，Standby ResourceManager会从Statestore目录获取Application相关信息，恢复数据。

Yarn 和 Tez 的关系

Hive on Tez作业信息需要Yarn提供TimeLine Server能力，以支持Hive任务展示应用程序的当前和历史状态，便于存储和检索。

说明

TimelineServer会将数据保存到内存数据库LevelDB中，占用大量内存，安装TimelineServer的节点内存至少需要预留30GB。

6.34.4 YARN 开源增强特性

任务优先级调度

在原生的YARN资源调度机制中，如果先提交的MapReduce Job长时间地占据整个Hadoop集群的资源，会使得后提交的Job一直处于等待状态，直到Running中的Job执行完并释放资源。

MRS集群提供了任务优先级调度机制。此机制允许用户定义不同优先级的Job，后启动的高优先级Job能够获取运行中的低优先级Job释放的资源；低优先级Job未启动的计算容器被挂起，直到高优先级Job完成并释放资源后，才被继续启动。

该特性使得业务能够更加灵活地控制自己的计算任务，从而达到更佳的集群资源利用率。

说明

容器可重用于任务优先级调度有冲突，若启用容器重用，资源会被持续占用，优先级调度将不起作用。

YARN 的权限控制

Hadoop YARN的权限机制是通过访问控制列表（ACL）实现的。按照不同用户授予不同权限控制，主要介绍下面两个部分：

- 集群运维管理员控制列表（Admin Acl）

  该功能主要用于指定YARN集群的运维管理员，其中，MRS集群管理员列表由参数“yarn.admin.acl”指定。集群运维管理员可以访问ResourceManager WebUI，还能操作NodeManager节点、队列、NodeLabel等，**但不能提交任务**。
- 队列访问控制列表（Queue Acl）

  为了方便管理集群中的用户，YARN将用户/用户组分成若干队列，并指定每个用户/用户组所属的队列。每个队列包含两种权限：提交应用程序权限和管理应用程序权限（比如终止任意应用程序）。

开源功能：

虽然目前YARN服务的用户层面上支持如下三种角色：

- 集群运维管理员
- 队列管理员
- 普通用户

但是当前开源YARN提供的WebUI/RestAPI/JavaAPI等接口上不会根据用户角色进行权限控制，任何用户都有权限访问应用和集群的信息，无法满足多租户场景下的隔离要求。

增强：

安全模式下，对开源YARN提供的WebUI/RestAPI/JavaAPI等接口上进行了权限管理上的增强，支持根据不同的用户角色，进行相应的权限控制。

各个角色对应的权限如下：

- 集群运维管理员：拥有在YARN集群上执行管理操作（如访问ResourceManager WebUI、刷新队列、设置NodeLabel、主备倒换等）的权限。

- 队列管理员：拥有在YARN集群上所管理队列的修改和查看权限。
- 普通用户：拥有在YARN集群上对自己提交应用的修改和查看权限。

自研超级调度器 Superior Scheduler 原理

Superior Scheduler是一个专门为Hadoop YARN分布式资源管理系统设计的调度引擎，是针对企业客户融合资源池，多租户的业务诉求而设计的高性能企业级调度器。

Superior Scheduler可实现开源调度器、Fair Scheduler以及Capacity Scheduler的所有功能。另外，相较于开源调度器，Superior Scheduler在企业级多租户调度策略、租户内多用户资源隔离和共享、调度性能、系统资源利用率和支持大集群扩展性方面都做了针对性的增强。设计的目标是让Superior Scheduler直接替代开源调度器。

类似于开源Fair Scheduler和Capacity Scheduler，Superior Scheduler通过YARN调度器插件接口与YARN Resource Manager组件进行交互，以提供资源调度功能。图6-137为其整体系统图。

图 6-137 Superior Scheduler 内部架构

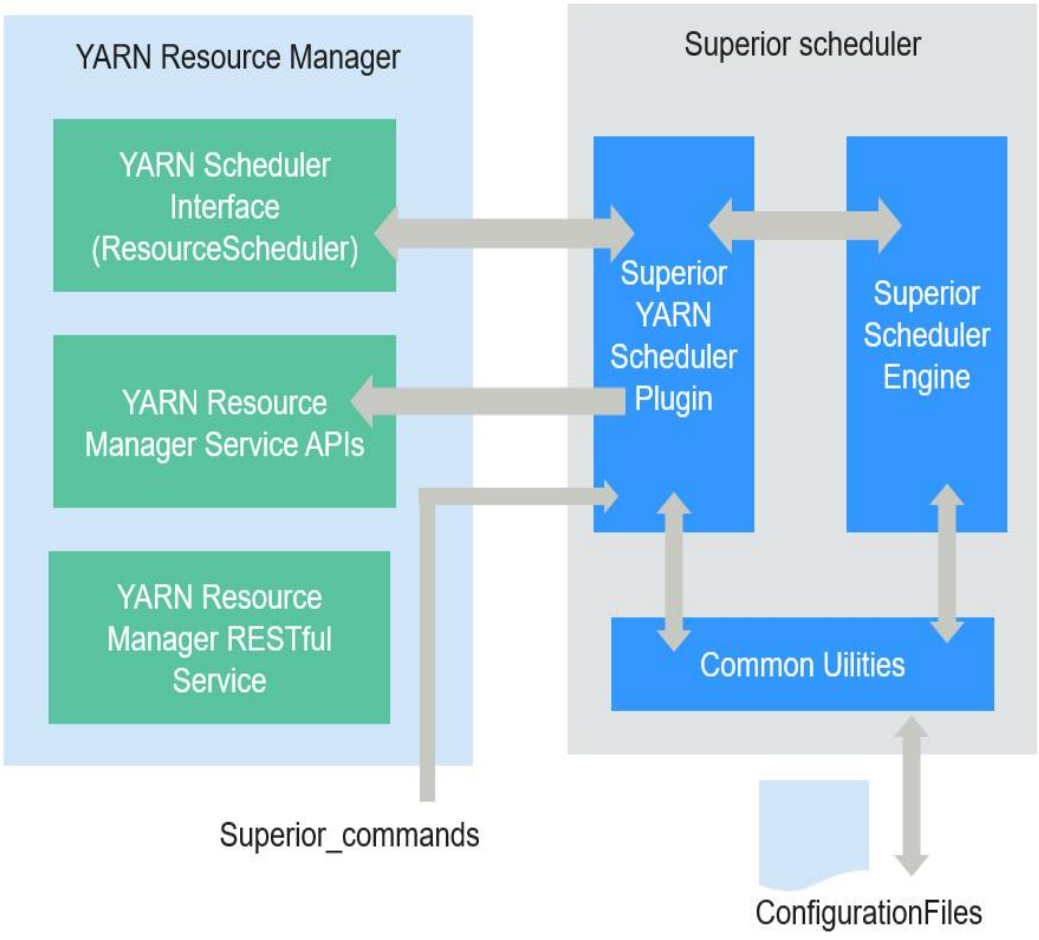


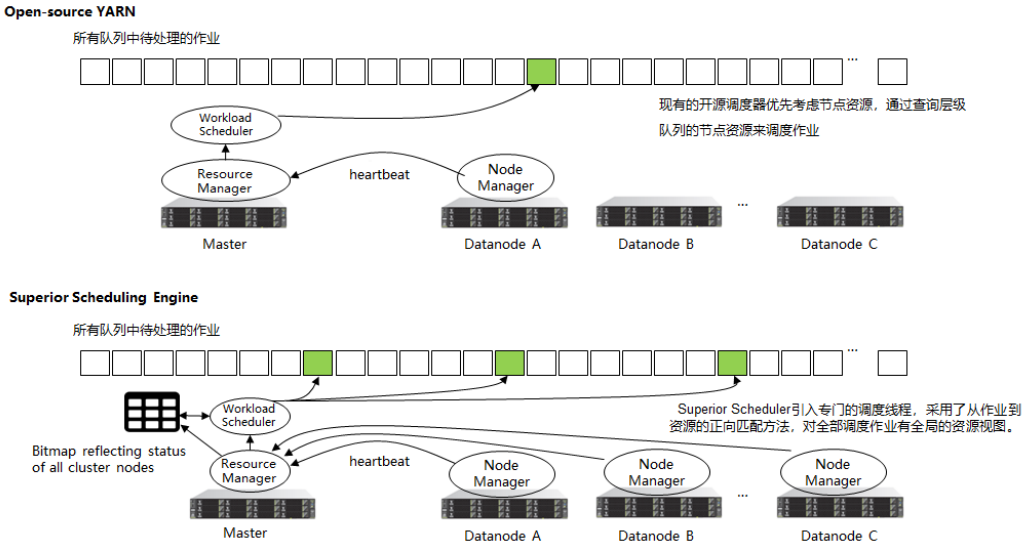
图6-137中，Superior Scheduler的主要模块如下：

- Superior Scheduler Engine：具有丰富调度策略的高性能调度器引擎。
- Superior YARN Scheduler Plugin：YARN Resource Manager和Superior Scheduler Engine之间的桥梁，负责同YARN Resource Manager交互。

在调度原理上，开源的调度器都是基于计算节点心跳驱动的资源反向匹配作业的调度机制。具体来讲，每个计算节点定期发送心跳到YARN的Resource Manager通知该节点状态并同时启动调度器为这个节点分配作业。这种调度机制把调度的周期同心跳结合在一起，当集群规模增大时，会遇到系统扩展性以及调度性能瓶颈。另外，因为采用了资源反向匹配作业的调度机制，开源调度器在调度精度上也有局限性，例如数据亲和性偏于随机，另外系统也无法支持基于负载的调度策略等。主要原因是调度器在选择作业时，缺乏全局的资源视图，很难做到好的选择。

Superior Scheduler内部采用了不同的调度机制。Superior Scheduler的调度器引入了专门的调度线程，把调度同心跳剥离开，避免了系统心跳风暴问题。另外，Superior Scheduler调度流程采用了从作业到资源的正向匹配方法，这样每个调度的作业都有全局的资源视图，可以很大地提高调度的精度。相比开源调度器，Superior Scheduler在系统吞吐量、利用率、数据亲和性等方面都有很大提升。

图 6-138 Superior Scheduler 性能对比



Superior Scheduler除了提高系统吞吐量和利用率，还提供了以下主要调度功能：

- 多资源池

  多资源池有助于在逻辑上划分集群资源并在多个租户/队列之间共享它们。资源池的划分可以基于异构的资源或完全按照应用资源隔离的诉求来划分。对于一个资源池，不同队列可配置进一步的策略。

- 每个资源池多租户调度（reserve、min、share、max）

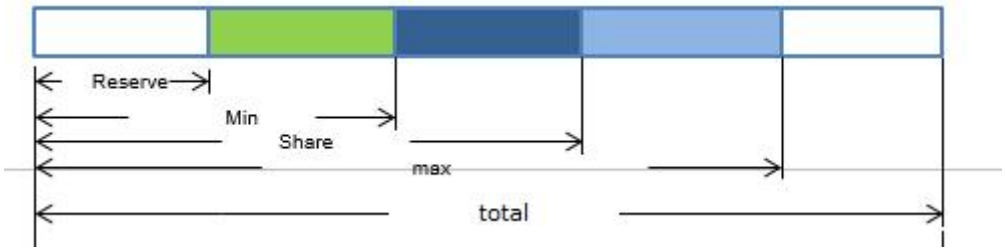
  Superior Scheduler提供了灵活的层级多租户调度策略。并允许针对不同的资源池可以访问的租户/队列，配置不同策略，如下所示。

表 6-30 策略描述

策略名称	描述
reserve	预留租户资源。即使租户没有作业，其他租户也不能使用该预留的资源。其值可以是百分比或绝对值。如果两者都配置，调度系统动态计算转换为资源绝对值，并取两者的最大值。缺省的reserve值为0。相对于定义一个专用资源池并指定具体机器的方式，reserve的策略可以认为提供了一种灵活的浮动预留功能，由于并不限定具体的机器，可以提高计算的数据亲和性，也不会受具体机器故障的影响。
min	具有抢占支持的最低保证资源。其他租户可以使用这部分资源，但是本租户享有优先使用权。其值可以是百分比或绝对值。如果两者都配置，调度系统动态计算转换为资源绝对值，并取两者的最大值。缺省值是0。
share	不支持抢占的共享资源。本租户要使用这部分资源时，需要等待其他租户完成作业并释放资源。其值是百分比或绝对值。
max	允许的最大资源数量。租户无法获得比允许的最大资源多的资源。其值是百分比或绝对值。如果两者都配置，调度系统动态计算转换为资源绝对值，并取两者最大值。缺省值不受限制。

租户资源分配策略示意图，如图6-139所示。

图 6-139 策略示意图



说明

其中“total”表示总资源，不是调度策略。

同开源的调度器相比，Superior Scheduler同时提供了租户级百分比和绝对值的混配策略，可以很好地适应各种灵活的企业级租户资源调度诉求。例如，用户可以在一级租户提供最大绝对值的资源保障，这样租户的资源不会因为集群的规模改变而受影响。但在下层的子租户之间，可以提供百分比的分配策略，这样可以尽可能提升一级租户内的资源利用率。

- 异构和多维资源调度

  Superior Scheduler除支持CPU和内存资源的调度外，还支持扩展以下功能：

  - 节点标签可用于识别不同节点的多维属性，可以根据这些标签进行调度。
  - 资源池可用于对同一类别的资源进行分组并分配给特定的租户/队列。
- 租户内多用户公平调度

在叶子租户里，多个用户可以使用相同的队列来提交作业。相比开源调度器，Superior Scheduler可以支持在同一租户内灵活配置不同用户的资源共享策略。例如可以为VIP用户配置更多的资源访问权重。

- 数据位置感知调度

  Superior Scheduler采用“从作业到节点的调度策略”，即尝试在可用节点之间调度给定的作业，使得所选节点适合于给定作业。通过这样做，调度器将具有集群和数据的整体视图。如果有机会使任务更接近数据，则保证了本地化。而开源调度器采用“从节点到作业的调度策略”，在给定节点中尝试匹配适当的作业。

- Container调度时动态资源预留

  在异构和多样化的计算环境中，一些container需要更多的资源或多种资源，例如Spark作业可能需要更大的内存。当这些container与其他需要较少资源的container竞争时，可能没有机会在合理的时间内获得所需的资源而处于饥饿状态。由于开源的调度器是基于资源反向匹配作业的调度方式，会为这些作业盲目地进行资源预留以防进入饥饿状态。这就导致了系统资源的整体浪费。Superior Scheduler与开源特性的不同之处在于：

  - 基于需求的匹配：由于Superior Scheduler采用“从作业到节点的调度”，能够选择合适的节点来预留资源提升这些特殊container的启动时间，并避免浪费。
  - 租户重新平衡：启用预留逻辑时，开源调度器并不遵循配置的共享策略。Superior Scheduler采取不同的方法。在每个调度周期中，Superior Scheduler将遍历租户，并尝试基于多租户策略重新达到平衡，且尝试满足所有策略（reserve，min，share等），以便可以释放预留的资源，将可用资源流向不同租户下的其他本应得到资源的container。

- 动态队列状态控制（Open/Closed/Active/Inactive）

  支持多个队列状态，有助于MRS集群管理员操作和维护多个租户。

  - Open状态（Open/Closed）：如果是Open（默认）状态，将接受提交到此队列的应用程序，如果是Closed状态，则不接受任何应用程序。
  - Active状态（Active/Inactive）：如果处于Active（默认）状态，租户内的应用程序是可以被调度和分配资源。如果处于Inactive状态则不会进行调度。

- 应用等待原因

  如果应用程序尚未启动，则提供作业等待原因信息。

Superior Scheduler和YARN开源调度器做了对比分析，如表6-31所示：

表 6-31 对比分析

领域	YARN开源调度器	Superior Scheduler
多租户调度	在同构集群上，只能选择容量调度器（Capacity Scheduler）或公平调度器（Fair Scheduler）两者之一，且集群当前不支持公平调度器（Fair Scheduler）。容量调度器只支持百分比方式配置，而公平调度器只支持绝对值方式。	• 支持**异构集群**和**多资源池**。 • 支持**预留**，以保证直接访问资源。

领域	YARN开源调度器	Superior Scheduler
数据位置感知调度	从节点到作业的调度策略导致降低数据本地化命中率，潜在影响应用的执行性能。	**从作业到节点的调度策略。**可具有更精确的数据位置感知，数据本地化调度的作业命中率比较高。
基于机器负载的均衡调度	不支持	**Superior Scheduler在调度时考虑机器的负载和资源分配情况，做到均衡调度。**
租户内多用户公平调度	不支持	租户内用户的公平调度，支持关键字default、others。
作业等待原因	不支持	作业等待原因信息可显示为什么作业需等待。

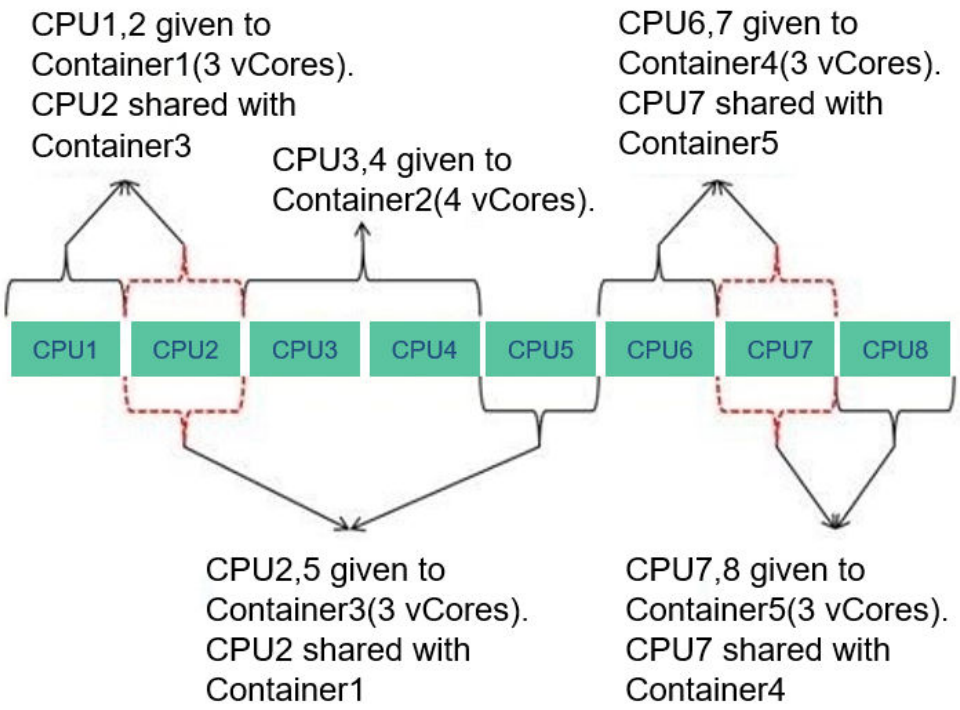
综上所述，Superior Scheduler是一个高性能调度器，拥有丰富的调度策略，在功能、性能、资源利用率和扩展性方面都优于Capacity Scheduler。

支持 CPU 硬隔离

YARN无法严格控制每个container使用的CPU资源。在使用CPU子系统时，container可能会超额占用资源。此时使用CPUset控制资源分配。

为了解决这个问题，CPU将会被严格按照虚拟核和物理核的比例分配至各个container。如果container需要一整个物理核，则分配给它一整个物理核。若container只需要部分物理核，则可能发生几个container共享同一个物理核的情况。下图为CPU配额示例，假定虚拟核和物理核的比例为2:1。

图 6-140 CPU 配额

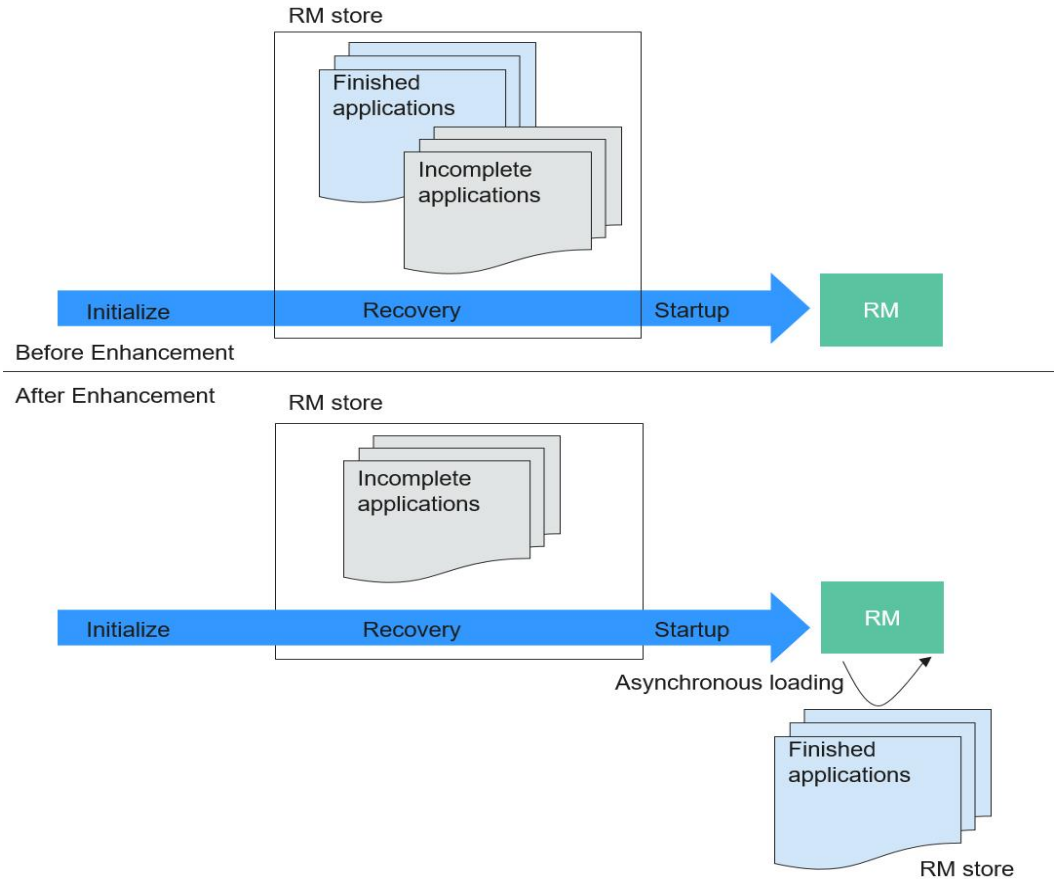


YARN 开源增强特性：重启性能优化

一般情况下，RM恢复会获取正在运行和已完成的应用。而大量的已完成的应用可能导致RM启动过慢、HA切换/重启耗时过长等问题。

为了加速RM的启动，现在优先获取未完成的应用列表，再启动RM。此时，已完成的应用会在一个后台异步线程中继续恢复。下图展示了RM的启动恢复流程。

图 6-141 RM 启动恢复流程



6.35 ZooKeeper

6.35.1 ZooKeeper 基本原理

ZooKeeper 简介

ZooKeeper是一个分布式、高可用性的协调服务。在大数据产品中主要提供两个功能：

- 帮助系统避免单点故障，建立可靠的应用程序。
- 提供分布式协作服务和维护配置信息。

更多关于ZooKeeper组件操作指导，请参考**使用ZooKeeper**。

ZooKeeper 结构

ZooKeeper集群中的节点分为三种角色：Leader、Follower和Observer，其结构和相互关系如图6-142所示。通常来说，需要在集群中配置奇数个（2N+1）ZooKeeper服务，至少（N+1）个投票才能成功的执行写操作。

图 6-142 ZooKeeper 结构

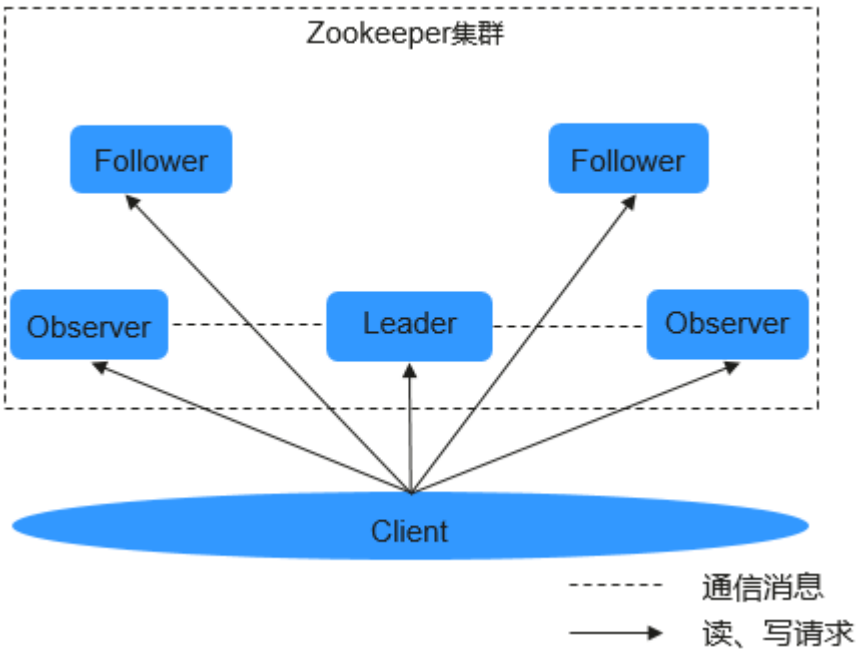


图6-142中各部分的功能说明如表6-32所示。

表 6-32 结构图说明

名称	描述
Leader	在ZooKeeper集群中只有一个节点作为集群的Leader，由各Follower通过ZooKeeper Atomic Broadcast(ZAB)协议选举产生，主要负责接收和协调所有写请求，并把写入的信息同步到Follower和Observer。
Follower	Follower的功能有两个： • 每个Follower都作为Leader的储备，当Leader故障时重新选举Leader，避免单点故障。 • 处理读请求，并配合Leader一起进行写请求处理。
Observer	Observer不参与选举和写请求的投票，只负责处理读请求、并向Leader转发写请求，避免系统处理能力浪费。
Client	ZooKeeper集群的客户端，对ZooKeeper集群进行读写操作。例如HBase可以作为ZooKeeper集群的客户端，利用ZooKeeper集群的仲裁功能，控制其HMaster的“Active”和“Standby”状态。

如果集群启用了安全服务，在连接ZooKeeper时需要进行身份认证，认证方式有以下两种：

- keytab方式：需要从MRS集群管理员处获取一个“人机”用户，用于登录MRS平台并通过认证，并且获取到该用户的keytab文件。
- 票据方式：从MRS集群管理员处获取一个“人机”用户，用于后续的安全登录，开启Kerberos服务的renewable和forwardable开关并且设置票据刷新周期，开启成功后重启kerberos及相关组件。

  说明

  - 默认情况下，用户的密码有效期是90天，所以获取的keytab文件的有效期是90天。
  - Kerberos服务的renewable、forwardable开关和票据刷新周期的设置在Kerberos服务的配置页面的“系统”标签下，票据刷新周期的修改可以根据实际情况修改“kdc_renew_lifetime”和“kdc_max_renewable_life”的值。

ZooKeeper 原理

- **写请求**
  a. Follower或Observer接收到写请求后，转发给Leader。
  b. Leader协调各Follower，通过投票机制决定是否接受该写请求。
  c. 如果超过半数以上的Leader、Follower节点返回写入成功，那么Leader提交该请求并返回成功，否则返回失败。
  d. Follower或Observer返回写请求处理结果。
- **只读请求**
  客户端直接向Leader、Follower或Observer读取数据。

ZooKeeper 常见规格

ZooKeeper服务的常见系统规格如**ZooKeeper常见规格**所示。

表 6-33 ZooKeeper 常见规格

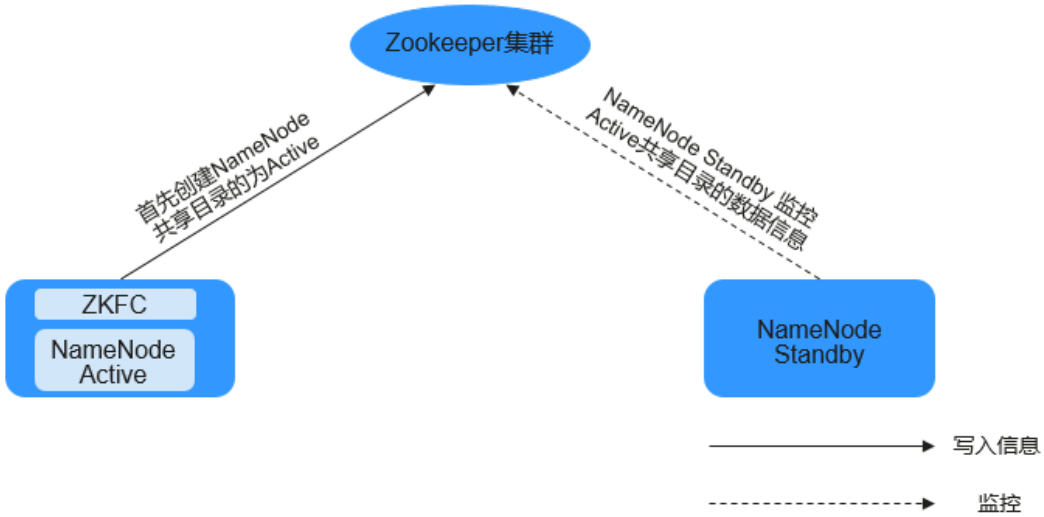
指标名称	规格	说明
单集群ZooKeeper最大实例数	9	ZooKeeper最大实例数
每个ZooKeeper实例，单个IP最大连接数	2000	-
每个ZooKeeper实例，最大连接总数	20000	-
默认参数情况下，最大ZNode数	2000000	ZNode数量过大会对服务稳定性造成影响，降低组件读写性能。 一般业务场景下建议ZNode数量在200w以内，如果集群仅部署了ClickHouse，ZNode数量可以扩大到600w以内。
单个ZNode大小	4MB	-

6.35.2 ZooKeeper 与其他组件的关系

ZooKeeper 和 HDFS 的关系

ZooKeeper与HDFS的关系如图6-143所示。

图 6-143 ZooKeeper 和 HDFS 的关系



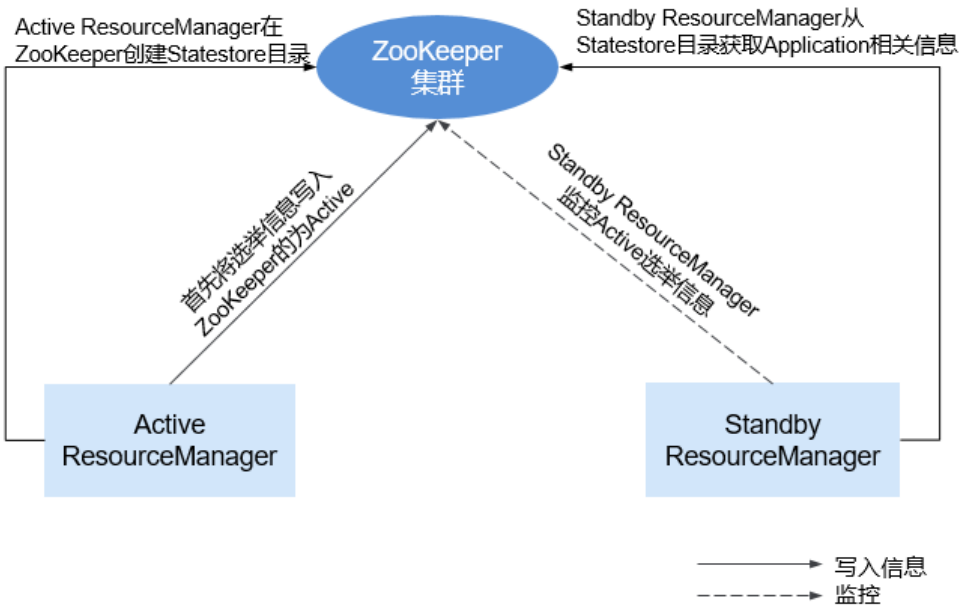
ZKFC（ZKFailoverController）作为一个ZooKeeper集群的客户端，用来监控NameNode的状态信息。ZKFC进程仅在部署了NameNode的节点中存在。HDFS NameNode的Active和Standby节点均部署有ZKFC进程。

1. HDFS NameNode的ZKFC连接到ZooKeeper，把主机名等信息保存到ZooKeeper中，即“/hadoop-ha”下的znode目录里。先创建znode目录的NameNode节点为主节点，另一个为备节点。HDFS NameNode Standby通过ZooKeeper定时读取NameNode信息。
2. 当主节点进程异常结束时，HDFS NameNode Standby通过ZooKeeper感知“/hadoop-ha”目录下发生了变化，NameNode会进行主备切换。

ZooKeeper 和 YARN 的关系

ZooKeeper与YARN的关系如图6-144所示。

图 6-144 ZooKeeper 与 YARN 的关系

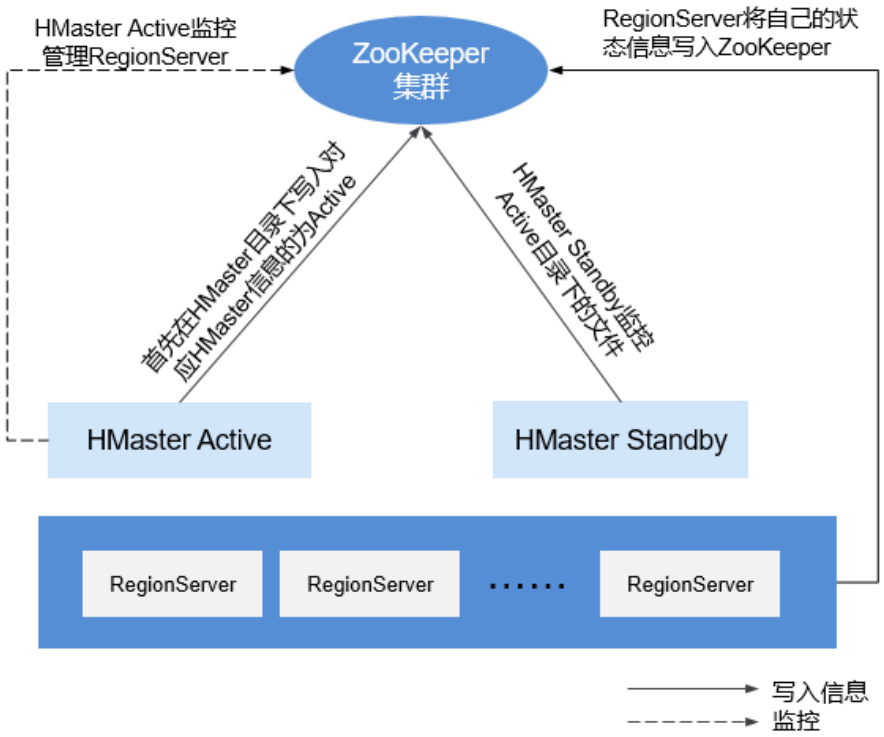


1. 在系统启动时，ResourceManager会尝试把选举信息写入ZooKeeper，第一个成功写入ZooKeeper的ResourceManager被选举为Active ResourceManager，另一个为Standby ResourceManager。Standby ResourceManager定时去ZooKeeper监控Active ResourceManager选举信息。
2. Active ResourceManager还会在ZooKeeper中创建Statestore目录，存储Application相关信息。当Active ResourceManager产生故障时，Standby ResourceManager会从Statestore目录获取Application相关信息，恢复数据。

ZooKeeper 和 HBase 的关系

ZooKeeper与HBase的关系如图6-145所示。

图 6-145 ZooKeeper 和 HBase 的关系

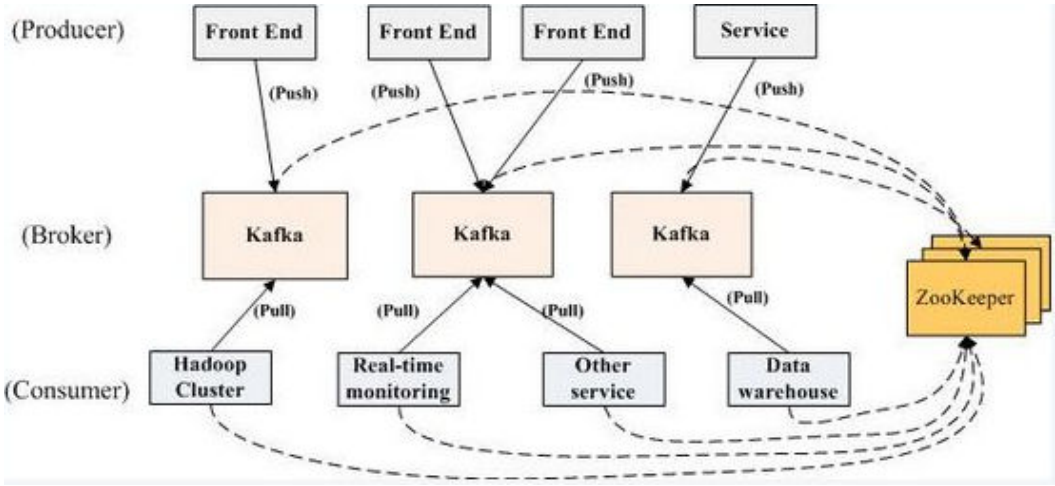


1. RegionServer以Ephemeral node的方式注册到ZooKeeper中。其中ZooKeeper存储HBase的如下信息：HBase元数据、HMaster地址。
2. HMaster通过ZooKeeper随时感知各个RegionServer的健康状况，以便进行控制管理。
3. HBase也可以部署多个HMaster，类似HDFS NameNode，当HMaster主节点出现故障时，HMaster备用节点会通过ZooKeeper获取主HMaster存储的整个HBase集群状态信息。即通过ZooKeeper实现避免HBase单点故障问题的问题。

ZooKeeper 和 Kafka 的配合关系

ZooKeeper与Kafka的关系如图 **ZooKeeper和Kafka的关系**所示。

图 6-146 ZooKeeper 和 Kafka 的关系



1. Broker端使用ZooKeeper用来注册broker信息，并进行partition leader选举。
2. Consumer端使用ZooKeeper用来注册consumer信息，其中包括consumer消费的partition列表等，同时也用来发现broker列表，并和partition leader建立socket连接，并获取消息。

6.35.3 ZooKeeper 开源增强特性

日志增强

安全模式下，Ephemeral node（临时节点）在session过期之后就会被系统删除，在审计日志中添加Ephemeral node被删除的审计日志，以便了解当时Ephemeral node的状态信息。

所有ZooKeeper客户端的操作都要在审计日志中添加Username。

从ZooKeeper客户端创建znode，其kerberos principal是“zkcli/hadoop.*<系统域名>*@*<系统域名>*”。

例如打开日志<ZOO_LOG_DIR>/zookeeper_audit.log，内容如下：

```
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
```

输出显示了在审计日志中添加了ZooKeeper客户端用户“zkcli/hadoop.hadoop.com@HADOOP.COM”的日志。

ZooKeeper中的用户详情：

在ZooKeeper中，不同的认证方案使用不同的凭证作为用户。 基于认证供应商的要求，任何参数都可以被认为是用户。

示例：

- SAMLAuthenticationProvider使用客户端主体作为用户。
- X509AuthenticationProvider使用客户端证书作为用户。
- IAuthenticationProvider使用客户端IP作为用户。
- 自定义认证提供程序实现org.apache.zookeeper.server.auth.ExtAuthenticationProvider.getUserName（String）方法以获取用户名。 如果没有实现，从认证提供程序实例获取用户名将被跳过。

ZooKeeper 开源增强特性：ZooKeeper SSL 通信（Netty 连接）

ZooKeeper设计最初含有Nio包，且不能较好的支持3.5版本后的SSL。为了解决这个问题，Netty被加入到ZooKeeper中。所以如果用户需要使用SSL，启用Netty并设置Server端和Client端的以下参数。

开源的服务端只支持简单的文本密码，这可能导致相关安全问题。为此在服务端将不再使用此类文本密码。

- Client端
  a. 将“zkCli.sh/zkEnv.sh”文件中的参数“-Dzookeeper.client.secure”设置为“true”以在Client端使用安全通信。之后客户端可以连接服务端的secureClientPort。
  b. 通过设置“zkCli.sh/zkEnv.sh”文件中的以下参数配置客户端环境。

参数	描述
-Dzookeeper.clientCnxnSocket	用于客户端的Netty通信。 默认值： "org.apache.zookeeper.ClientCnxnSocketNetty"
-Dzookeeper.ssl.keyStore.location	keystore文件路径。
-Dzookeeper.ssl.keyStore.password	加密密码。
-Dzookeeper.ssl.trustStore.location	truststore文件路径。
-Dzookeeper.ssl.trustStore.password	加密密码。
-Dzookeeper.config.crypt.class	用于加密密码的解密。
-Dzookeeper.ssl.password.encrypted	默认值：false 当keystore和truststore的密码为加密密码时设置为true。

参数	描述
-Dzookeeper.ssl.enabled.protocols	通过配置此参数定义SSL协议以适用于SSL上下文。
-Dzookeeper.ssl.exclude.cipher.ext	通过配置此参数定义SSL上下文中应排除的密码列表，之间以逗号间隔。

说明

以上参数须在“zkCli.sh/zkEnv.sh”文件内设置。

- Server端
  a. 在文件“zoo.cfg”中将SSL端口参数“secureClientPort”设置为“3381”。
  b. 在server端将文件“zoo.cfg”中的参数“zookeeper.serverCnxnFactory”设置为“org.apache.zookeeper.server.NettyServerCnxnFactory”。
  c. 设置文件zoo.cfg（路径：“zookeeper/conf/zoo.cfg”）中的以下参数来配置服务端环境。

参数	描述
ssl.keyStore.location	keystore.jks文件路径。
ssl.keyStore.password	加密密码。
ssl.trustStore.location	truststore文件路径。
ssl.trustStore.password	加密密码。
config.crypt.class	用于加密密码的解密。
ssl.keyStore.password.encrypted	默认值：false 设置为true时可使用加密密码。
ssl.trustStore.password.encrypted	默认值：false 设置为true时可使用加密密码。
ssl.enabled.protocols	通过配置此参数定义SSL协议以适用于SSL上下文。
ssl.exclude.cipher.ext	通过配置此参数定义SSL上下文中应排除的密码列表，之间以逗号间隔。

  d. 启动ZKserver，然后将安全客户端连接到安全端口。
- 凭证
  ZooKeeper上Client和Server之间的凭证由X509AuthenticationProvider执行。根据以下参数指定服务端证书及信任客户端证书，并通过这些证书初始化X509AuthenticationProvider。
  – zookeeper.ssl.keyStore.location

- zookeeper.ssl.keyStore.password
- zookeeper.ssl.trustStore.location
- zookeeper.ssl.trustStore.password

说明

若用户不想使用ZooKeeper的默认机制，可根据所需配置不同的ZooKeeper信任机制。

7 产品功能

7.1 作业管理

MRS作业管理为用户提供向集群提交作业的入口，支持包括MapReduce、Spark、HQL和SparkSQL等类型的作业。

结合华为云数据治理中心DataArts Studio，提供一站式的大数据协同开发环境、全托管的大数据调度能力，帮助用户快速构建大数据处理中心。

通过数据治理中心DataArts Studio，用户可以先在线开发调试MRS HQL/SparkSQL脚本、拖拽式地开发MRS作业，完成MRS与其他20多种异构数据源之间的数据迁移和数据集成；通过强大的作业调度与灵活的监控告警，轻松管理数据作业运维。

目前MRS集群支持在线创建如下几种类型的作业：

- MapReduce：提供快速并行处理大量数据的能力，是一种分布式数据处理模式和执行环境，MRS支持提交MapReduce Jar程序。
- Spark：基于内存进行计算的分布式计算框架，MRS支持提交SparkSubmit、Spark Script和Spark SQL作业。
  - SparkSubmit：提交Spark Jar和Spark Python程序，运行Spark Application计算和处理用户数据。
  - SparkScript：提交SparkScript脚本，批量执行Spark SQL语句。
  - Spark SQL：使用Spark提供的类似SQL的Spark SQL语句，实时查询和分析用户数据。
- Hive：建立在Hadoop基础上的开源的数据仓库。MRS支持提交HiveScript脚本和直接执行Hive SQL语句。
- Flink：提供一个分布式大数据处理引擎，可对有限数据流和无限数据流进行有状态计算。
- HadoopStreaming：HadoopStreaming作业像普通Hadoop作业一样，除了可以指定输入和输出的HDFS路径的参数外，它还可以指定mapper和reducer的可执行程序。

更多关于MRS集群作业提交操作指导，请参考**提交MRS作业**。

7.2 元数据管理

当创建MRS集群选择部署Hive和Ranger组件时，MRS提供多种元数据存储方式，您可以根据自身需要进行选择：

- 本地元数据：元数据存储于集群内的本地GaussDB中，当集群删除时元数据同时被删除，如需保存元数据，需提前前往数据库手动保存元数据。
- 外置数据连接：MRS集群创建完成后，可选择关联与当前集群同一虚拟私有云和子网的RDS服务中的PostgresDB或MySQL数据库，元数据将存储于关联的数据库中，不会随当前集群的删除而删除，多个MRS集群可共享同一份元数据。

说明

Hive组件可选元数据存储方式功能在MRS 1.9.x及之后版本支持。

更多关于MRS集群元数据外置操作指导，请参考**管理MRS集群元数据**。

7.3 企业项目管理

企业项目是一种云资源管理方式。企业管理提供面向企业客户的云上资源管理、人员管理、权限管理、财务管理等综合管理服务。区别于管理控制台独立操控、配置云产品的方式，企业管理控制台以面向企业资源管理为出发点，帮助企业以公司、部门、项目等分级管理方式实现企业云上的人员、资源、权限、财务的管理。

MRS支持已开通企业项目服务的用户在创建集群时为集群配置对应的项目，然后使用企业项目管理对MRS上的资源进行分组管理：

- 支持用户为多个资源进行分组管理。
- 支持用户查看企业项目下的资源信息、消费明细。
- 支持用户对企业项目级别的访问权限控制。
- 支持用户分企业项目查看具体的财务信息，包括订单、消费汇总、消费明细等。

说明

若MRS集群与VPC不在同一个企业项目中，用户需要在IAM视图添加VPC查看权限后方可查看VPC及集群相关信息。

7.4 多租户资源管理

特性简介

现代企业的数据集群在向集中化和云化方向发展，企业级大数据集群需要满足：

- 不同用户在集群上运行不同类型的应用和作业（分析、查询、流处理等），同时存放不同类型和格式的数据。
- 部分用户（例如银行、政府单位等）对数据安全非常关注，不接受将自己的数据与其他用户放在一起。

这给大数据集群带来了以下挑战：

- 合理地分配和调度资源，以支持多种应用和作业在集群上平稳运行。

- 对不同的用户进行严格的访问控制，以保证数据和业务的安全。

多租户将大数据集群的资源隔离成一个个资源集合，彼此互不干扰，用户通过“租用”需要的资源集合，来运行应用和作业，并存放数据。在大数据集群上可以存在多个资源集合来支持多个用户的不同需求。

因此，MRS大数据集群提供了完整的企业级大数据多租户解决方案。多租户是MRS大数据集群中的多个资源集合（每个资源集合是一个租户），具有分配和调度资源（资源包括计算资源和存储资源）的能力。

更多关于MRS集群租户资源配置与管理操作，请参考**管理MRS集群租户**。

特性优势

- 合理配置和隔离资源

  租户之间的资源是隔离的，一个租户对资源的使用不影响其他租户，保证了每个租户根据业务需求去配置相关的资源，可提高资源利用效率。

- 测量和统计资源消费

  系统资源以租户为单位进行计划和分配，租户是系统资源的申请者和消费者，其资源消费能够被测量和统计。

- 保证数据安全和访问安全

  多租户场景下，分开存放不同租户的数据，以保证数据安全；控制用户对租户资源的访问权限，以保证访问安全。

调度器增强

多租户根据调度器类型分为开源的Capacity调度器和华为自主研发的增强型Superior调度器。

为满足企业需求，克服YARN社区在调度上遇到的挑战与困难，华为自主研发的Superior调度器，不仅集合了当前Capacity调度器与Fair调度器的优点，还做了以下增强：

- 增强资源共享策略

  Superior调度器支持队列层级，在同集群集成开源调度器的特性，并基于可配置策略进一步共享资源。针对实例，MRS集群管理员可通过Superior调度器为队列同时配置绝对值或百分比的资源策略计划。Superior调度器的资源共享策略将YARN的标签调度增强为资源池特性，YARN集群中的节点可根据容量或业务类型不同，进行分组以使队列更有效地利用资源。

- 基于租户的资源预留策略

  部分租户可能在某些时间中运行关键任务，租户所需的资源应保证可用。Superior调度器构建了支持资源预留策略的机制，在这些租户队列运行的任务可立即获取到预留资源，以保证计划的关键任务可正常执行。

- 租户和资源池的用户公平共享

  Superior调度器提供了队列内用户间共享资源的配置能力。每个租户中可能存在不同权重的用户，高权重用户可能需要更多共享资源。

- 大集群环境下的调度性能优势

  Superior调度器接收到各个NodeManager上报的心跳信息，并将资源信息保存在内存中，使得调度器能够全局掌控集群的资源使用情况。Superior调度器采用了push调度模型，令调度更加精确、高效，大大提高了大集群下的资源使用率。另外，Superior调度器在NodeManager心跳间隔较大的情况下，调度性能依然优异，不牺牲调度性能，也能避免大集群环境下的“心跳风暴”。

- 优先策略

  当某个服务在获取所有可用资源后还无法满足最小资源的要求，则会发生优先抢占。抢占功能默认关闭。

7.5 组件 WebUI 便捷访问

大数据组件都有自己的WebUI页面管理自身系统，但是由于网络隔离的原因，用户并不能很简便地访问到该页面。

例如访问HDFS的WebUI页面，传统的操作方法是需要用户创建ECS，使用ECS远程登录组件的UI，这使得组件的页面UI访问很是繁琐，对于很多初次接触大数据的用户很不友好。

MRS提供了基于弹性公网IP来便捷访问组件WebUI的安全通道，并且比用户自己绑定弹性公网IP更便捷，只需界面鼠标操作，即可简化原先用户需要自己登录虚拟私有云添加安全组规则，获取公网IP等步骤，减少了用户操作步骤。

分析集群Hadoop、Spark、HBase、Hue及流式集群等相关组件，都可以在Manager上找到组件页面入口，快速访问。

更多关于MRS集群内托管的开源组件WebUI界面登录操作指导，请参考**访问MRS集群上托管的开源组件Web页面**。

7.6 节点自定义引导操作

特性简介

MRS提供标准的云上弹性大数据集群，目前可安装部署包括Hadoop、Spark等大数据组件。当前标准的云上大数据集群不能满足所有用户需求，例如如下几种场景：

- 通用的操作系统配置不能满足实际数据处理需求，例如需调大系统最大连接数。
- 需要安装自身业务所需的软件工具或运行环境，例如需安装Gradle、业务需要依赖R语言包。
- 根据自身业务对大数据组件包做修改，例如对Hadoop或Spark安装包做修改。
- 需要安装其他MRS还未支持的大数据组件。

对于上述定制化的场景，可以选择登录到每个节点上手动操作，之后每扩容一个新节点，再执行一次同样的操作，操作相对繁琐，也容易出错。同时手动执行记录不便追溯，不能实现“按需创建、创建成功后即处理数据”的目标。

因此，MRS提供了自定义引导操作，在启动集群组件前（或后）可以在指定的节点上执行脚本。用户可以通过引导操作来完成安装MRS还没支持的第三方软件，修改集群运行环境等自定义操作。如果集群扩容，选择执行引导操作，则引导操作也会以相同方式在新增节点上执行。MRS会使用root用户执行用户指定的脚本，脚本内部可以通过**su - *xxx***命令切换用户。

客户价值

MRS提供了自定义引导操作，用户可以灵活、便捷地配置自己的专属集群，自定义安装软件。

更多关于MRS集群内节点引导操作配置，请参考**配置MRS集群节点引导操作**。

7.7 集群管理

7.7.1 集群生命周期管理

MRS支持集群的生命周期管理包括创建集群和删除集群。

- 创建集群：支持用户定制集群的类型、组件范围、各类型的节点数、虚拟机规格、可用区、VPC网络、认证信息，MRS将为用户自动创建一个符合配置的集群，全程无需用户参与；同时支持用户在集群中运行自定义内容；支持快速创建多应用场景集群，比如创建Hadoop分析集群、HBase集群、Kafka集群。大数据平台同时支持部署异构集群，在集群中存在不同规格的虚机，允许CPU类型，硬盘容量，硬盘类型，内存大小灵活组合。在集群中支持多种虚机规格混合使用。
- 删除集群：当按需计费的集群不再需要时（包括集群中的数据和配置），用户可以选择删除集群，MRS会将集群相关的资源全部删除。
- 续订：目前MRS提供按需和包年/包月购买方式，按需是每小时扣费，如果余额不足将导致欠费，而包年/包月集群需要在时长用完前续费。如果您未能在按需集群欠费后或者包年/包月集群到期后续费，华为云不会立即停止您的业务，订单转入保留期，此时集群将终止服务，数据仍然保留。
- 退订：已经购买包周期集群的客户，在集群资源到期之前，如果不需要该集群资源，可以在MRS上对已订购包周期的产品进行集群资源退订。

购买集群

通过在MRS服务管理面，客户可以按需或者包年包月购买MRS集群，通过选择集群所建的区域及使用的云资源规格，一键式购买适合企业业务的MRS集群。MRS服务会根据用户选择的集群类型、版本和节点规格，帮助客户自动完成华为云企业级大数据平台的安装部署和参数调优。

MRS服务为客户提供完全可控的大数据集群，客户在创建时可设置虚拟机的登录方式（密码或者密钥对），所创建的MRS集群资源完全归客户所用。

MRS集群类型包括分析集群、流式集群和混合集群。

- 分析集群：用来做离线数据分析，提供Hadoop体系的组件。
- 流式集群：用来做流处理任务，提供流式处理组件。
- 混合集群：既可以用来做离线数据分析，又可以用来做流处理任务，提供Hadoop体系的组件和流式处理组件。
- 自定义：根据业务需求，可以灵活搭配所需组件（MRS 3.x及后续版本）。

MRS集群节点类型包括Master节点、Core节点和Task节点。

- Master节点：集群中的管理节点。分布式系统的Master进程和Manager以及数据库均部署在该节点；该类型节点不可扩容。该类型节点的处理能力决定了整个集群的管理上限，MRS服务支持将Master节点规格提高，以支持更大集群的管理。
- Core节点：支持存储和计算两种目标的节点，可扩容、缩容。因承载数据存储功能，因此在缩容时，为保证数据不丢失，有较多限制，无法进行弹性伸缩。
- Task节点：仅用于计算的节点，可扩容、缩容。因只承载计算任务，因此可以进行弹性伸缩。

MRS购买集群方式支持自定义购买集群和快速购买集群两种。

- 自定义购买集群：自定义购买可以灵活地选择计费模式、配置项，针对不同的应用场景，可以选择不同规格的弹性云服务器，全方位贴合您的业务诉求。
- 快速购买集群：用户可以根据应用场景，快速购买对应配置的集群，提高了配置效率，更加方便快捷。当前支持快速购买Hadoop分析集群、HBase集群、Kafka集群、ClickHouse集群、实时分析集群。
  - Hadoop分析集群：Hadoop分析集群完全使用开源Hadoop生态，采用YARN管理集群资源，提供Hive、Spark离线大规模分布式数据存储和计算，SparkStreaming、Flink流式数据计算，Presto交互式查询，Tez有向无环图的分布式计算框等Hadoop生态圈的组件，进行海量数据分析与查询。
  - HBase集群：HBase集群使用Hadoop和HBase组件提供一个稳定可靠，性能优异、可伸缩、面向列的分布式云存储系统，适用于海量数据存储以及分布式计算的场景，用户可以利用HBase搭建起TB至PB级数据规模的存储系统，对数据轻松进行过滤分析，毫秒级得到响应，快速发现数据价值。
  - Kafka集群：Kafka集群使用Kafka和Storm组件提供一个开源高吞吐量，可扩展性的消息系统。广泛用于日志收集、监控数据聚合等场景，实现高效的流式数据采集，实时数据处理存储等。
  - ClickHouse集群：ClickHouse集群是一个用于联机分析的列式数据库管理系统，具有压缩率和极速查询性能。广泛用于互联网广告、App和Web流量、电信、金融、物联网等众多领域。
  - 实时分析集群：实时分析集群使用Hadoop、Kafka、Flink和ClickHouse组件提供一个海量的数据采集、数据的实时分析和查询的系统。

删除集群

MRS服务支持用户在不需要大数据集群时执行删除集群操作，集群删除后，所有大数据使用的相关云资源都会同时被释放。删除集群前，建议完成数据搬迁或者备份，确认集群无任何业务运行或者集群异常且经运维分析无法继续提供服务时再执行集群删除操作。对于数据存放在云硬盘EVS或直通盘的大数据集群，集群删除后，数据也随之删除，强烈建议您慎重选择删除集群。

7.7.2 集群在线扩缩容

大数据集群的处理能力通常可以通过增加集群的节点数来横向扩展，当集群规模不符合业务要求时，用户可以通过该功能进行集群节点规模的调整，进行扩容或者缩容；在缩容节点时，MRS会智能地选择负载最少或者迁移数据量最小节点，并且在缩容过程中，缩容节点不再接收新的任务，正在执行的任务继续执行，同时将该节点数据拷贝至其他节点，该节点进入退服状态，当该节点任务长时间运行无法结束时，会迁移至其他节点运行，最大限度地减少对集群业务的影响。

扩容集群

目前支持扩容集群Core节点或Task节点，用户可通过增加节点数量处理业务峰值负载。

MRS集群节点扩容中和扩容后对现有集群的业务没有影响，数据类组件扩容后引起的数据倾斜需手动进行数据均衡操作。

更多关于MRS集群扩容操作指导及注意事项，请参考**扩容MRS集群**。

包周期集群扩容

当用户购买了MRS包周期集群后，在订购的周期之内，用户的业务增长超过预期时，就会出现超出包周期订单规模外的扩容诉求。MRS服务支持包周期集群扩容能力，做到了在轻松帮助您完成扩容的前提下，让您继续享受着包周期的优惠。

您只需要打开MRS服务页面，通过界面操作便可扩容出您需要的节点数。整个扩容过程无需后台人工介入，只需几分钟，即可完美解决您遇到的日益上涨的业务数据压力。

缩容集群

用户可以根据业务需求量，通过简单地缩减Core节点或者Task节点，对集群进行缩容，以使MRS拥有更优的存储、计算能力，降低运维成本。用户执行MRS集群缩容后，MRS服务将根据节点已安装的服务类型自动选择可以缩容的节点。

Core节点在缩容的时候，会对原节点上的数据进行迁移。业务上如果对数据位置做了缓存，客户端自动刷新位置信息可能会影响时延。缩容节点可能会影响部分HBase on HDFS数据的第一次访问响应时长，可以重启HBase或者对相关的表执行Disable/Enable操作来避免。

Task节点本身不存储集群数据，属于计算节点，不存在节点数据迁移的问题。

更多关于MRS集群缩容操作指导及注意事项，请参考**缩容MRS集群**。

7.7.3 创建 Task 节点

特性简介

MRS集群支持创建Task节点，只作为计算节点，不存放持久化的数据，是实现弹性伸缩的基础。

客户价值

在MRS服务只作为计算资源的场景下，使用Task节点可以节省成本，并可以更加方便快捷地对集群节点进行扩缩容，满足用户对集群计算能力随时增减的需求。

用户场景

当集群数据量变化不大而集群业务处理能力需求变化比较大，临时需要增大业务量时，可选择添加Task节点。

- 临时业务量增大，如年底报表处理。
- 需要在短时间内处理大量的任务，如一些紧急分析任务。

7.7.4 自动弹性伸缩

特性简介

随着企业的数据越来越多，越来越多的企业选择使用Spark/Hive等技术来进行分析，由于数据量大，任务处理繁重，资源消耗较高，因此使用成本也越来越高。当前并不是每个企业在每时每刻在进行分析，而一般是在一天的一个时间段内进行分析汇总，因此MRS提供了弹性伸缩能力，可以自动在业务在繁忙时申请额外资源，业务不繁忙时释放闲置资源，让用户按需使用，尽可能地帮助客户降低使用成本，聚焦核心业务。

在大数据应用，尤其是周期性的数据分析处理场景中，需要根据业务数据的周期变化，动态调整集群计算资源以满足业务需要。MRS的弹性伸缩规则功能支持根据集群负载对集群进行弹性伸缩。此外，如果数据量为周期有规律的变化，并且希望在数据量变化前提前完成集群的扩缩容，可以使用MRS的资源计划特性。

MRS服务支持规则和时间计划两种弹性伸缩的策略：

- 弹性伸缩规则：根据集群实时负载对Task节点数量进行调整，数据量变化后触发扩缩容，有一定的延后性。
- 资源计划：若数据量变化存在周期性规律，则可通过资源计划在数据量变化前提前完成集群的扩缩容，避免出现增加或减少资源的延后。

弹性伸缩规则与资源计划均可触发弹性伸缩，两者既可同时配置也可单独配置。资源计划与基于负载的弹性伸缩规则叠加使用可以使得集群节点的弹性更好，足以应对偶尔超出预期的数据峰值出现。

当某些业务场景要求在集群扩缩容之后，根据节点数量的变化对资源分配或业务逻辑进行更改时，手动扩缩容的场景需要用户登录集群节点进行操作。对于弹性伸缩场景，MRS支持通过自定义弹性伸缩自动化脚本来解决。自动化脚本可以在弹性伸缩前后执行相应操作，自动适应业务负载的变化，免去了人工操作。同时，自动化脚本给用户实现个性需求提供了途径，完全自定义的脚本与多个可选的执行时机基本可以满足用户的各项需求，使弹性伸缩更具灵活性。

客户价值

MRS的自动弹性伸缩可以帮助用户实现以下价值。

- 降低使用成本

  部分企业并不是时刻都在进行批量分析，例如一般情况下数据持续接入，而到了特定时间段（例如凌晨3点）进行批量分析，可能仅需要消耗2小时。

  MRS提供的弹性伸缩能力，可以帮助用户在进行批量分析操作时，将分析节点扩容到指定规模，而计算完毕后，则自动释放计算节点，尽可能地降低使用成本。

- 平衡突发查询

  大数据集群上，由于有大量的数据，企业会经常面临临时的分析任务，例如支撑企业决策的临时数据报表等，都会导致对于资源的消耗在极短时间内剧增。MRS提供的弹性伸缩能力，可以在突发大数据分析时，及时补充计算节点，避免因为计算能力不足，导致业务宕机。用户无需手动购买额外资源，当突发事件结束后，MRS会自动判断缩容时机，自动完成缩容。

- 聚焦核心业务

  大数据作为二次开发平台，开发人员时常难以判断具体的资源消耗，由于查询分析的条件复杂性（例如全局排序，过滤，合并等）以及数据的复杂性（例如增量数据的不确定性等），都会导致预估计算量难以进行，而使用弹性伸缩能力，可以让业务人员专注于业务开发，无需分心再做各种资源评估。

更多关于MRS集群Task节点弹性伸缩配置，请参考**MRS集群Task节点弹性伸缩**。

7.7.5 节点隔离

当用户发现某个主机出现异常或故障，无法提供服务或影响集群整体性能时，可以临时将主机从集群可用节点排除，使客户端访问其他可用的正常节点。

在为MRS集群安装补丁的场景中，也支持排除指定节点不安装补丁。

说明

隔离主机仅支持隔离MRS集群内的非管理节点。

主机隔离后该主机上的所有角色实例将被停止，且不能对主机及主机上的所有实例进行启动、停止和配置等操作。

主机隔离后无法统计并显示该主机硬件和主机上实例的监控状态及指标数据。

更多关于MRS集群节点隔离操作，请参考**隔离MRS集群节点**。

7.7.6 升级 Master 节点规格

MRS大数据集群采用Manager实现集群的管理，而管理集群的相关服务，如HDFS存储系统的NameNode，Yarn资源管理的ResourceManager，以及MRS的Manager管理服务都部署在集群的Master节点上。

随着新业务的上线，集群规模不断扩大，Master节点承担的管理负荷也越来越高，企业用户面临CPU负载过高，内存使用率超过阈值的问题。通常自建大数据集群需要完成数据搬迁，采购升级节点硬件配置实现Master规格提升，而MRS服务借助云服务的优势，实现一键式Master节点升级，并在升级过程中通过Master节点的主备HA保证已有业务的不间断，方便快捷帮助用户解决主节点规格升级问题。

Master节点具体升级操作请参见**升级Master节点规格**。

7.7.7 节点标签管理

标签是集群/节点的标识，为集群/节点添加标签，可以方便用户识别和管理拥有的集群/节点资源。MRS服务通过与标签管理服务（TMS）关联，可以让拥有大量云资源的用户，通过给云资源打标签，快速查找具有同一标签属性的云资源，进行统一检视、修改、删除等管理操作，方便用户对大数据集群及其他相关云资源的统一管理。

您可以在创建集群时添加标签，也可以在集群创建完成后，在集群的详情页添加标签，您最多可以给集群添加10个标签。

节点的标签仅支持在节点创建完成后，在节点的标签页添加标签，您最多可以给节点添加10个标签。

更多关于MRS集群节点标签操作，请参考**添加MRS集群节点标签**。

7.8 集群运维

告警管理

MRS可以实时监控大数据集群，通过告警和事件可以识别系统健康状态。同时MRS也支持用户自定义配置监控与告警阈值用于关注各指标的健康情况，当监控数据达到告警阈值，系统将会触发一条告警信息。

MRS还可以与华为云消息通知服务(SMN)的消息服务系统对接，将告警信息通过短信或者邮件等形式推送给用户，具体介绍请参见**集群状态消息通知**。

补丁管理

MRS集群支持补丁操作，会及时发布开源大数据组件的补丁。用户能够在MRS集群管理页面上查看到运行集群相关的补丁发布信息，包括其修复问题的详细说明及影响场

景，用户可以根据业务运行情况自行选择是否安装补丁。补丁安装过程是一键式操作，无需人工干预，通过滚动安装，补丁升级不会停止业务，保障用户集群长期可用。

MRS服务可以展示详细的补丁安装过程，支持补丁的卸载和失败回滚。

运维支撑

MRS集群的资源完全属于用户，通常情况下，当集群出现问题需要运维人员支撑时，运维人员无法直接访问该集群。为了更好的服务客户，MRS提供两种方式来减少定位问题时的信息传递：

- 日志共享：用户可以在MRS页面发起日志共享，选择日志范围共享给运维人员，以便运维人员在不接触集群的情况下帮助定位问题。
- 运维授权：MRS服务提供运维授权功能，用户在使用MRS集群过程中，发生问题可以在MRS页面发起运维授权，由运维人员帮助用户快速定位问题，用户可以随时收回该授权。

健康检查

MRS为用户提供界面化的系统运行环境自动检查服务，帮助用户实现一键式系统运行健康度巡检和审计，保障系统的正常运行，降低系统运维成本。用户查看检查结果后，还可导出检查报告用于存档及问题分析。

7.9 集群状态消息通知

特性简介

大数据集群运行过程中经常会进行如下操作：

- 大数据集群变更，比如扩容、缩容集群。
- 业务数据量突然变化，集群触发弹性伸缩。
- 相关业务结束，需要终止大数据集群等。

用户想要及时得知这些操作是否执行成功，以及当集群出现大数据服务不可用，或节点故障时，用户希望不用频繁登录集群查看，就可以及时地收到告警通知。MRS联合消息通知服务(SMN)，可以将以上信息主动地通知到用户的手机及邮箱，让维护更加省心省力。

客户价值

配置消息通知后，可以实时给用户发送MRS集群健康状态，用户可以通过手机短信或邮箱实时接收到MRS集群变更及组件告警信息。MRS可以帮助用户轻松运维，实时监控，实时发送告警，操作灵活，使大数据业务部署更加省心省力。

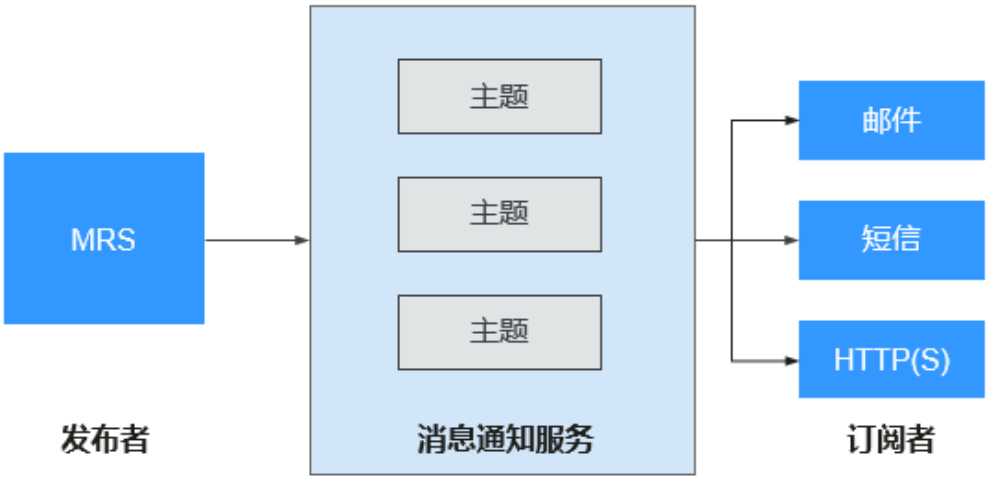
特性描述

MRS联合消息通知服务(SMN)，采用主题订阅模型，提供一对多的消息订阅以及通知功能，能够实现一站式集成多种推送通知方式。

首先，作为主题拥有者，可以先创建一个主题，并对主题设置访问控制权限来决定哪些发布者和订阅者可以通过该主题进行交流。MRS将集群消息发送至您有权限发布消

息的主题，然后所有订阅了该主题的订阅者（可以是手机短信、邮箱等）都将收到集群变更以及组件告警的消息。

图 7-1 MRS 集群状态消息通知



更多关于MRS集群告警事件消息通知操作，请参考**配置MRS集群告警事件消息通知**。

7.10 MRS 安全增强

MRS作为一个海量数据管理和分析的平台，具备高安全性。MRS主要从以下几个方面保障用户的数据和业务运行安全。

- 网络隔离

  整个系统部署在公有云上的虚拟私有云中，提供隔离的网络环境，保证集群的业务、管理的安全性。结合虚拟私有云的子网划分、路由控制、安全组等功能，为用户提供高安全、高可靠的网络隔离环境。

- 资源隔离

  MRS服务支持资源专属区内部署，专属区内物理资源隔离，用户可以在专属区内灵活地组合计算存储资源，包括专属计算资源+共享存储资源、共享计算资源+专属存储资源、专属计算资源+专属存储资源。

- 主机安全

  MRS支持与公有云安全服务集成，支持漏洞扫描、安全防护、应用防火墙、堡垒机、网页防篡改等。针对操作系统和端口部分，华为云提供如下安全措施：

  - 操作系统内核安全加固
  - 操作系统权限控制
  - 操作系统端口管理

- 应用安全

  通过如下措施保证大数据业务正常运行：

  - 身份鉴别和认证
  - Web应用安全
  - 访问控制

  - 审计安全
  - 密码安全
- 数据安全

  针对海量用户数据，提供如下措施保障客户数据的机密性、完整性和可用性。
  - 容灾：MRS支持将数据备份到OBS（对象存储服务）中，支持跨区域的高可靠性。
  - 备份：MRS支持针对DBService、NameNode、LDAP的元数据备份和对HDFS、HBase的业务数据备份。
- 数据完整性

  通过数据校验，保证数据在存储、传输过程中的数据完整性。
  - 用户数据保存在HDFS上，HDFS默认采用CRC32C校验数据的正确性。
  - HDFS的DataNode节点负责存储校验数据，如果发现客户端传递过来的数据有异常（不完整）就上报异常给客户端，让客户端重新写入数据。
  - 客户端从DataNode读数据的时候会同步检查数据是否完整，如果发现数据不完整，尝试从其它的DataNode节点上读取数据。
- 数据保密性

  MRS分布式文件系统在Apache Hadoop版本基础上，提供对文件内容的加密存储功能，避免敏感数据明文存储，提升数据安全性。业务应用只需对指定的敏感数据进行加密，加解密过程业务完全不感知。在文件系统数据加密基础上，Hive实现表级加密，HBase实现列族级加密，在创建表时指定采用的加密算法，即可实现对敏感数据的加密存储。

  从数据的存储加密、访问控制来保障用户数据的保密性。
  - HBase支持将业务数据存储到HDFS前进行压缩处理，且用户可以配置AES和SM4（也称为SMS4）算法加密存储。
  - 各组件支持本地数据目录访问权限设置，无权限用户禁止访问数据。
  - 所有集群内部用户信息提供密文存储。
- 安全认证
  - 基于用户和角色的认证统一体系，遵从账户/角色RBAC（Role-Based Access Control）模型，实现通过角色进行权限管理，对用户进行批量授权管理。
  - 支持安全协议Kerberos，MRS使用LDAP作为账户管理系统，并通过Kerberos对账户信息进行安全认证。
  - 提供单点登录，统一了MRS系统用户和组件用户的管理及认证。
  - 对登录Manager的用户进行审计。

7.11 MRS 可靠性增强

MRS在基于Apache Hadoop开源软件的基础上，在主要业务部件的可靠性、性能调优等方面进行了优化和提升。

系统可靠性

- 管理节点均实现HA

  Hadoop开源版本的数据、计算节点已经是按照分布式系统进行设计的，单节点故障不影响系统整体运行；而以集中模式运作的管理节点可能出现的单点故障，就成为整个系统可靠性的短板。

MRS对所有业务组件的管理节点都提供了类似的双机的机制，包括Manager、HDFS NameNode、HiveServer、HBase HMaster、YARN ResourceManager、KerberosServer、LdapServer等，全部采用主备或负荷分担配置，有效避免了单点故障场景对系统可靠性的影响。

- 异常场景下的可靠性保证

  通过可靠性分析方法，梳理软件、硬件异常场景下的处理措施，提升系统的可靠性。

  - 保障意外掉电时的数据可靠性，不论是单节点意外掉电，还是整个集群意外断电，恢复供电后系统能够正常恢复业务，除非硬盘介质损坏，否则关键数据不会丢失。
  - 硬盘亚健康检测和故障处理，对业务不造成实际影响。
  - 自动处理文件系统的故障，自动恢复受影响的业务。
  - 自动处理进程和节点的故障，自动恢复受影响的业务。
  - 自动处理网络故障，自动恢复受影响的业务。

- 数据备份与恢复

  为应对数据丢失或损坏对用户业务造成不利影响，在异常情况下快速恢复系统，MRS根据用户业务的需要提供全量备份、增量备份和恢复功能。

  - 自动备份

    MRS对集群管理系统Manager上的数据提供自动备份功能，根据制定的备份策略可自动备份集群上的数据，包括LdapServer、DBService的数据。

  - 手动备份

    在系统进行扩容、打补丁等重大操作前，需要通过手动备份集群管理系统的数据，以便在系统故障时，恢复集群管理系统功能。

    为进一步提供系统的可靠性，在将Manager、HBase上的数据备份到第三方服务器时，也需要通过手动备份。

节点可靠性

- 操作系统健康状态监控

  周期采集操作系统硬件资源使用率数据，包括CPU、内存、硬盘、网络等资源的使用率状态。

- 进程健康状态监控

  MRS提供业务实例的状态以及业务实例进程的健康指标的检查，能够让用户第一时间感知进程健康状态。

- 硬盘故障的自动处理

  MRS对开源版本进行了增强，可以监控各节点上的硬盘以及文件系统状态。如果出现异常，立即将相关分区移出存储池；如果硬盘恢复正常（通常是因为用户更换了新硬盘），也会将新硬盘重新加入业务运作。这样极大简化了维护人员的工作，更换故障硬盘可以在线完成；同时用户可以设置热备盘，从而极大缩减了故障硬盘的修复时间，有利于提高系统的可靠性。

- 节点磁盘LVM配置

  MRS支持将多个磁盘配置成LVM（Logic Volume Management），多个磁盘规划成一个逻辑卷组。配置成LVM可以避免各磁盘间使用不均的问题，保持各个磁盘间均匀使用在HDFS和Kafka等能够利用多磁盘能力的组件上尤其重要。并且LVM可以支持磁盘扩容时不需要重新挂载，避免了业务中断。

数据可靠性

MRS可以利用弹性云服务器ECS提供的反亲和节点组能力，结合Hadoop的机架感知能力，将数据冗余到多个物理宿主机上，避免物理硬件的失效造成数据的失效。

8 安全

8.1 责任共担

华为云秉承“将公司对网络和业务安全性保障的责任置于公司的商业利益之上”。针对层出不穷的云安全挑战和无孔不入的云安全威胁与攻击，华为云在遵从法律法规业界标准的基础上，以安全生态圈为护城河，依托华为独有的软硬件优势，构建面向不同区域和行业的完善云服务安全保障体系。

与传统的本地数据中心相比，云计算的运营方和使用方分离，提供了更好的灵活性和控制力，有效降低了客户的运营负担。正因如此，云的安全性无法由一方完全承担，云安全工作需要华为云与您共同努力，如图8-1所示。

- **华为云**：无论在任何云服务类别下，华为云都会承担基础设施的安全责任，包括安全性、合规性。该基础设施由华为云提供的物理数据中心（计算、存储、网络等）、虚拟化平台及云服务组成。在PaaS、SaaS场景下，华为云也会基于控制原则承担所提供服务或组件的安全配置、漏洞修复、安全防护和入侵检测等职责。
- **客户**：无论在任何云服务类别下，客户数据资产的所有权和控制权都不会转移。在未经授权的情况，华为云承诺不触碰客户数据，客户的内容数据、身份和权限都需要客户自身看护，这包括确保云上内容的合法合规，使用安全的凭证（如强口令、多因子认证）并妥善管理，同时监控内容安全事件和账号异常行为并及时响应。

图 8-1 华为云安全责任共担模型

云安全责任基于控制权，以可见、可用作为前提。在客户上云的过程中，资产（例如设备、硬件、软件、介质、虚拟机、操作系统、数据等）由客户完全控制向客户与华为云共同控制转变，这也就意味着客户需要承担的责任取决于客户所选取的云服务。如图8-1所示，客户可以基于自身的业务需求选择不同的云服务类别（例如IaaS、PaaS、SaaS服务）。不同的云服务类别中，每个组件的控制权不同，这也导致了华为云与客户的责任关系不同。

- 在On-prem场景下，由于客户享有对硬件、软件和数据等资产的全部控制权，因此客户应当对所有组件的安全性负责。
- 在IaaS场景下，客户控制着除基础设施外的所有组件，因此客户需要做好除基础设施外的所有组件的安全工作，例如应用自身的合法合规性、开发设计安全，以及相关组件（如中间件、数据库和操作系统）的漏洞修复、配置安全、安全防护方案等。
- 在PaaS场景下，客户除了对自身部署的应用负责，也要做好自身控制的中间件、数据库、网络控制的安全配置和策略工作。
- 在SaaS场景下，客户对客户内容、账号和权限具有控制权，客户需要做好自身内容的保护以及合法合规、账号和权限的配置和保护等。

8.2 资产识别与管理

通信安全授权

MRS服务通过管理控制台为用户发放、管理和使用大数据组件，大数据组件部署在用户的VPC内部，MRS管理控制台需要直接访问部署在用户VPC内的大数据组件时需要开通相应的安全组规则，而开通相应的安全组规则需要获取用户授权，此授权过程称为通信安全授权。

通信安全授权需要在创建集群时完成，如图8-2所示。

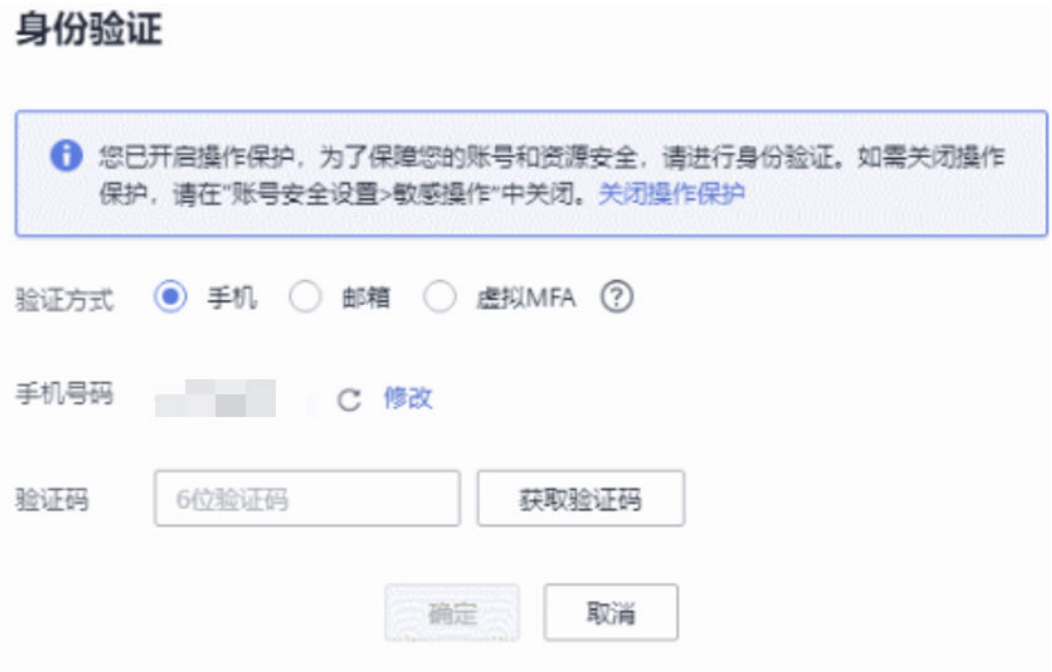
图 8-2 通信安全授权

建议用户在使用大数据组件时，仅给可信的IP地址放开安全组访问权限。谨慎使用0.0.0.0/0作为安全组源地址。

敏感操作保护

MRS支持对敏感操作进行保护，若用户已开启敏感操作保护（请参见IAM服务的**敏感操作**），则输入选择的对应验证方式获取的验证码进行验证（如**图8-3**所示），避免误操作带来的风险和损失。

图 8-3 身份验证



8.3 身份认证与访问控制

身份认证

MRS支持安全协议Kerberos，使用LDAP作为账户管理系统，并通过Kerberos服务对账户信息进行安全认证。

Kerberos安全认证原理和认证机制具体介绍请参见**安全认证原理和认证机制**。

访问控制

MRS提供两种访问控制权限模型：基于角色的权限控制和基于策略的权限控制，详情请参见**权限模型**。

- **基于角色的权限控制**

  MRS基于用户和角色的认证统一体系，遵从账户/角色RBAC（Role-Based Access Control）模型，实现通过角色进行权限管理，对用户进行批量授权管理，同时提供单点登录能力，统一了系统用户和组件用户的管理及认证。具体机制详情描述请参见**权限机制**。

- **基于策略的权限控制**
  - Ranger鉴权

    MRS提供了基于Ranger的鉴权方案，对于MRS安全集群，默认启用了Ranger鉴权；对于安装了Ranger服务的普通集群，Ranger可以支持基于OS用户进行组件资源的权限控制。

    Ranger鉴权的具体策略请参见**鉴权策略**。
  - OBS存算分离细粒度鉴权

    对于OBS存算分离集群，如果您想对OBS上的资源进行细粒度的权限控制，可以通过MRS提供的基于IAM委托的细粒度权限控制方案进行配置，请参见**配置MRS多用户访问OBS细粒度权限**。

8.4 数据保护技术

数据完整性

通过数据校验，保证数据在存储、传输过程中的数据完整性。

MRS的用户数据保存在HDFS中，HDFS默认采用CRC32C算法校验数据的正确性，同时也支持CRC32校验算法，CRC32C校验速度快于CRC32。HDFS的DataNode节点负责存储校验数据，如果发现客户端传递过来的数据有异常（不完整）就上报给客户端，让客户端重新写入数据。客户端从DataNode读数据的时候也一样要检查数据是否完整，如果发现数据不完整，会尝试从其他的DataNode节点上读取数据。

数据保密性

MRS分布式文件系统在Apache Hadoop版本基础上提供对文件内容的加密存储功能，避免敏感数据明文存储，提升数据安全性。

业务应用只需对指定的敏感数据进行加密，加解密过程业务完全不感知。在文件系统数据加密基础上，Hive服务支持列加密（参见**使用Hive列加密功能**），可以在创建表时指定加密算法实现对敏感数据的加密存储。HBase支持加密HFile和WAL内容，用户可以配置AES和SM4（也称为SMS4）算法进行数据的加密存储（参见**加密HFile和WAL内容**）。

数据传输安全性

在MRS集群中，Web通道访问支持HTTPS加密；RPC通信支持SASL认证，并可配置对称密钥的方式进行加密。

组件级别的传输加密配置如下所示：

- HDFS配置传输加密：请参见**配置HDFS数据传输加密**。
- Kafka配置传输加密：请参见**配置Kafka数据传输加密**。
- Flume配置传输加密：请参见**配置Flume加密传输**。
- Flink配置传输加密：请参见**认证和加密**章节的加密传输操作指导。

数据容灾与备份

- 容灾：MRS支持将数据备份到对象存储服务（OBS）中，支持跨区域的高可靠性。
- 备份：MRS支持针对OMS、Kafka、DBService、NameNode等组件的元数据备份和对HDFS、HBase、Hive等组件的业务数据备份。

关于备份能力详细的内容介绍请参见**备份恢复简介**。

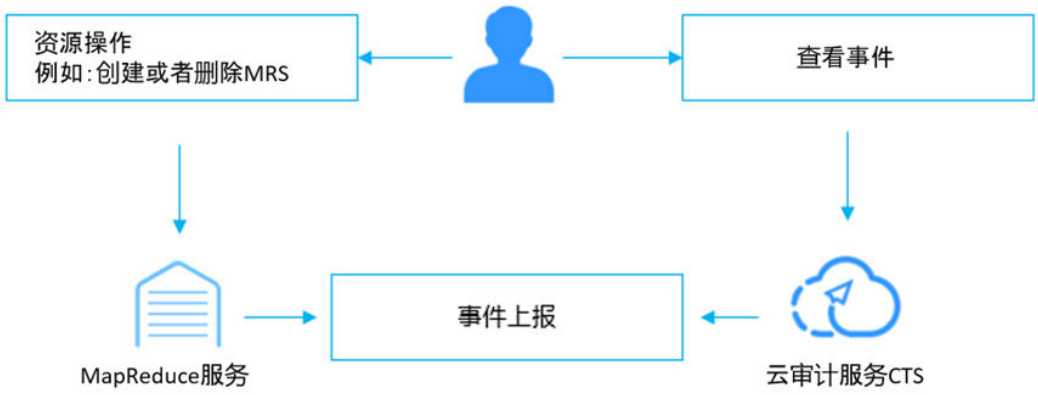
8.5 审计与日志

审计

MRS服务在管理控制台上的操作日志，例如创建或删除MRS集群的日志记录，通过云审计服务（Cloud Trace Service，CTS）实现。CTS是华为云安全解决方案中专业的日志审计服务，提供对各种云资源操作记录的收集、存储和查询功能，可用于支撑安全分析、合规审计、资源跟踪和问题定位等常见应用场景。

用户开通云审计服务并创建和配置追踪器后，CTS可记录MRS的管理事件和数据事件用于审计。

图 8-4 CTS 记录 MRS 事件



同时FusionInsight Manager也提供了审计功能，可以记录用户对集群Manager页面操作信息。管理员可通过“审计”页面查看用户在Manager上的历史操作记录，用于安全事件中定位问题原因及划分责任。审计管理页面介绍请参见**审计管理页面概述**。Manager的审计日志默认保存在数据库中，如果长期保留可能引起数据目录的磁盘空间不足问题，管理员如果需要将审计日志保存到其他归档服务器，可以在FusionInsight Manager设置转储参数及时自动转储，便于管理审计日志信息。审计日志转储操作指导请参见**配置审计日志转储**。

日志

MRS集群所有组件日志（如HDFS服务全部日志）支持通过主机接入的方式对接云日志服务。云日志服务（LTS）用于收集来自主机和云服务的日志数据，通过海量日志数据的分析与处理，可以将云服务和应用程序的可用性和性能最大化，为您提供实时、高效、安全的日志处理能力，帮助您快速高效地进行实时决策分析、设备运维管理、用户业务趋势分析等。具体对接指导请参见**MRS服务如何对接云日志服务**。

同时FusionInsight Manager支持在线检索并显示组件的日志内容，用于问题定位等其他日志查看场景，详细操作指导请参见**在线检索日志**。FusionInsight Manager支持批量导出各个服务角色所有实例生成的日志，无需手工登录单个节点获取，详细操作指导请参见**下载日志**。

8.6 服务韧性

跨AZ容灾部署能力

MRS服务管理面提供双集群跨AZ容灾能力，即在另一个可用区（跨AZ）部署一个同构的MRS灾备集群。

如果生产集群所处的地理位置发生自然灾害，或者集群内部出现了故障从而导致生产集群无法正常对外提供读写服务，那么灾备集群可以切换为生产集群，从而保障业务连续性。

8.7 监控安全风险

MRS的Manager界面提供集群级别的监控能力，帮助用户监控集群中大数据组件和节点的健康状态，同时提供告警通知能力，用户可以实时掌握MRS集群的各项指标、健康度。

MRS支持将集群中所有部署角色的节点，按管理节点、控制节点和数据节点进行分类，分别计算关键主机监控指标在每类节点上的变化趋势，并在报表中按用户自定义的周期显示分布曲线图。MRS集群指标监控采用周期性监控，历史监控平均周期约为5分钟。

用户可在MRS管理控制台或者Manager界面中查看集群整体的资源概况。

更多详情请参见**查看和定制集群监控指标**和**管理组件和主机监控**。

8.8 更新管理

密码更新

MRS支持集群内用户密码的更新，建议管理员定期修改密码，提高系统安全性。

密码更新指导如下所示：

- 修改系统用户密码：请参见**修改admin密码**和**修改操作系统用户密码**。
- **修改系统内部用户密码**
- **修改默认数据库用户密码**

证书更新

MRS集群的CA证书与HA证书均支持更换，如果用户需要将集群默认的证书更换成新的证书，可参考如下指导：

- CA证书用于组件客户端与服务端在通信过程中加密数据，实现安全通信。具体更换操作指导请参见**更换CA证书**。
- HA证书用于主备进程与高可用进程在通信过程中加密数据，实现安全通信。具体更换操作指导请参见**更换HA证书**。

8.9 安全加固

加固 Tomcat

在FusionInsight Manager使用过程中，针对Tomcat基于开源做了如下功能增强：

- 升级Tomcat版本为官方稳定版本。
- 设置应用程序之下的目录权限为500，对部分目录支持写权限。
- 系统软件安装完成后自动清除Tomcat安装包。
- 应用程序目录下针对工程禁用自动部署功能，只部署了web、cas和client三个工程。
- 禁用部分未使用的HTTP方法，防止被他人利用攻击。
- 更改Tomcat服务器默认shutdown端口号和命令，避免被黑客捕获利用关闭服务器，降低对服务器和应用的威胁。
- 出于安全考虑，更改“maxHttpHeaderSize”的取值，给服务器管理员更大的可控性，以控制客户端不正常的请求行为。
- 安装Tomcat后，修改Tomcat版本描述文件。
- 为了避免暴露Tomcat自身的信息，更改Connector的Server属性值，使攻击者不易获知服务器的相关信息。
- 控制Tomcat自身配置文件、可执行文件、日志目录、临时目录等文件和目录的权限。
- 关闭会话facade回收重用功能，避免请求泄漏风险。
- CookieProcessor使用LegacyCookieProcessor，避免cookie中的敏感数据泄漏。

加固 LDAP

MRS集群中针对LDAP做了如下功能增强：

- LDAP配置文件中管理员密码使用SHA加密，当升级openldap版本为2.4.39或更高时，主备LDAP节点服务自动采用SASL External机制进行数据同步，避免密码信息被非法获取。
- 集群中的LDAP服务默认支持SSLv3协议，可安全使用。当升级openldap版本为2.4.39或更高时，LDAP将自动使用TLS 1.0以上的协议通讯，避免未知的安全风险。

其它安全加固

其它一些集群安全加固指导请参见**安全加固**。

8.10 MRS 集群保留 JDK 说明

MRS集群是租户完全可控的大数据应用开发平台，用户基于平台开发业务后，将业务程序部署到大数据平台运行。由于需要具备开发调测能力，因此要在MRS集群中保留JDK。

此外，MRS集群功能中如下关键特性也强依赖JDK。

- HBase BulkLoad

  HBase BulkLoad支持用户自定义proto文件将数据文件中的字段导入HBase，该特性需要使用JDK将用户自定义的proto文件转换成Java文件，然后编译成Class文件运行。
- 组件进程堆栈信息采集

  MRS集群内角色或实例的堆栈信息采集功能依赖于JDK，具体参见“采集堆栈信息”章节。

须知

安全风险说明：JDK中包含javac、jmap、jdb等调测工具，攻击者可以利用调测工具调试业务进程，可能造成进程中敏感信息泄露。但是此类攻击需要攻击者拿到集群节点的Shell权限后才可以执行，MRS集群部署在VPC内，由安全组控制访问，故不将MRS集群暴露给不可信网络即可消解该风险。

9 约束与限制

使用MRS前，您需要认真阅读并了解以下使用限制。

MRS 集群创建限制

表 9-1 MRS 集群创建约束说明

限制项	说明
网络要求	● MRS集群必须创建在VPC子网内。 ● 创建MRS集群时，支持自动创建安全组，也可选择已有的安全组。 ● MRS集群使用的安全组请勿随意放开权限，避免被恶意访问。 ● 为保证集群运行正常，集群节点及集群网络可达的网络区域请勿规划使用以下IP：10.10.10.10、10.10.10.11、1.1.1.1。
浏览器	建议使用推荐的浏览器登录MRS管理界面。 ● Google Chrome：36.0及更高版本 ● Edge：随Windows操作系统更新。
数据存储	● MRS集群节点仅用于存储用户业务数据，非业务数据建议保存在对象存储服务或其他弹性云服务器中。 ● MRS集群节点仅用于运行MRS集群内服务，其他客户端应用程序、用户业务程序建议申请独立弹性云服务器部署。 ● 请根据业务需要规划集群节点的磁盘，如果需要存储大量业务数据，请及时增加云硬盘数量或存储空间，以防止存储空间不足影响节点正常运行。 ● MRS集群扩容（包含存储能力和计算能力）可通过增加Core节点或者Task节点的方式实现。
密码要求	MRS不会保存您设置的登录Master节点的初始密码，请您设置并保管好密码。为避免被恶意攻击，建议设置复杂度高的密码。

限制项	说明
技术支持	● 集群处于非人为异常状态时，可以联系技术支持人员，技术支持人员征得您同意后会请您提供密码，登录MRS集群进行问题排查。 ● 集群处于异常状态时，MRS仍然会收取集群费用。建议您及时联系技术支持人员处理集群异常。

MRS 集群运行限制

表 9-2 MRS 集群运行约束说明

限制项	说明
节点管理	● 当MRS集群中某一个Master节点关闭后，如果仍然使用集群执行作业任务或修改组件配置，在操作后必须先启动被关闭的Master节点，然后才能执行其他节点的关闭操作，否则会由于角色主备倒换导致数据丢失的风险。 ● 若MRS集群中节点已经被全部关闭，请按照节点关机顺序的倒序启动集群节点。
资源调度	当使用MRS集群过程中进行Capacity和Superior调度器切换时，系统只会完成调度器的切换，不保证队列配置同步。如果您需要配置同步，建议基于新的调度器重新配置。
存算分离	当集群已对接了OBS（存算分离或者冷热分离场景），若需要删除组件或者MRS集群，需要在删除组件或者集群后，手工将OBS上相关的业务数据进行删除。
组件特性	● MRS集群中HBase服务禁止开启MOB特性，使用该特性可能存在无法正常读取表数据和JVM Crash风险。 已存在的HBase表可在**hbase shell**命令行中执行以下命令查看表详情，排查表描述中是否包含MOB关键字，如果包含，需联系系统运维人员修改为非MOB表。 **desc '***表名***'** 例如，以下回显信息中“IS_MOB”值为“true”表示启用了HBase MOB特性： hbase:009:0> desc '*t3*' t3 COLUMN FAMILIES DESCRIPTION {NAME => 'd', MOB_THRESHOLD => '102400', VERSIONS => '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE => '0', BLOO MFILTER => 'ROW', IN_MEMORY => 'false', IS_MOB => 'true', COMPRESSION => 'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'} ● MRS集群中Spark及其他引擎禁止执行自读自写的SQL语句。 例如，以下SQL示例即为自读自写： insert overwrite table test select * from test; insert overwrite table test select a.* from test a join test2 b on a.name=b.name;

集群及组件高危操作（禁止）

MRS集群操作与组件使用过程中，应注意以下的禁用操作，防止集群不稳定导致业务运行异常。

表 9-3 MRS 集群禁用操作

操作类别	操作风险
• 在ECS服务管理控制台对MRS集群的节点进行关机、重启、删除、变更OS、重装OS和修改规格等操作。 • 删除MRS集群节点上已有的进程、安装的应用程序和文件。 • 删除MRS集群节点，集群节点丢失造成集群异常后依旧会收取费用，导致您的损失。	操作会导致MRS集群进入异常状态，影响MRS集群使用。
集群创建完成后，请勿随意删除或更改已使用的安全组。	操作可能导致集群异常，影响MRS集群的正常使用。
严禁删除ZooKeeper相关数据目录	ClickHouse/HDFS/Yarn/HBase/Hive等组件都依赖于ZooKeeper，在ZooKeeper中保存了元数据信息。删除ZooKeeper中相关数据目录将会影响上层组件的正常运行。
严禁JDBCServer主备节点频繁倒换	频繁主备倒换将导致业务中断。
严禁删除Phoenix系统表或系统表数据(SYSTEM.CATALOG、SYSTEM.STATS、SYSTEM.SEQUENCE、SYSTEM.FUNCTION)	删除系统表将导致无法正常进行业务操作。
严禁手动修改Hive元数据库的数据（hivemeta数据库）	修改Hive元数据可能会导致Hive数据解析错误，Hive无法正常提供服务。
禁止对Hive的元数据表手动进行**insert**和**update**操作	修改Hive元数据可能会导致Hive数据解析错误，Hive无法正常提供服务。
严禁修改Hive私有文件目录hdfs:///tmp/hive-scratch的权限	修改该目录权限可能会导致Hive服务不可用。
严禁修改Kafka配置文件中broker.id	修改Kafka配置文件中broker.id将会导致该节点数据失效。
严禁修改节点主机名	主机名修改后会导致该主机上相关实例和上层组件无法正常提供服务，且无法修复。
禁止使用私有镜像	该操作会导致MRS集群进入异常状态，影响MRS集群使用。
严禁修改MRS集群节点的默认DNS配置信息	修改DNS信息可能导致该节点对其他服务器域名解析出现异常，引发通信中断、集群异常等问题。

表9-4为MRS集群内各组件在操作与维护阶段应注意的高危操作。

表 9-4 MRS 集群高危操作

操作类别	操作名称	操作风险	规避措施
集群操作	随意修改omm用户下的文件目录或者文件权限	该操作会导致MRS集群服务不可用。	无。
	绑定弹性公网IP	该操作会将集群的manager所在的master节点暴露在公网，会增大来自互联网的网络攻击风险可能性。	请确认绑定的弹性公网IP为可信任的公网访问IP。
	开放集群22端口安全组规则	该操作会增大用户利用22端口进行漏洞攻击的风险。	针对开放的22端口进行设置安全组规则，只允许可信的IP可以访问该端口，入方向规则不推荐设置允许0.0.0.0可以访问。
	删除集群或删除集群数据	该操作会导致数据丢失。	删除前请务必再次确认该操作的必要性，同时要保证数据已完成备份。
	缩容集群	该操作会导致数据丢失。	缩容前请务必再次确认该操作的必要性，同时要保证数据已完成备份。
	卸载磁盘或格式化数据盘	该操作会导致数据丢失。	操作前请务必再次确认该操作的必要性，同时要保证数据已完成备份。
Manager操作	修改OMS密码	该操作会重启OMS各进程，影响集群的管理维护。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	导入证书	该操作会重启OMS进程和整个集群，影响集群的管理维护和业务。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	恢复OMS	该操作会重启Manager和整个集群，影响集群的管理维护和业务。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	修改日志级别	如果修改为DEBUG，会导致Manager运行速度明显降低。	修改前确认操作的必要性，并及时修改回默认设定。

操作类别	操作名称	操作风险	规避措施
	重启下层服务时勾选同时重启上层服务	该操作会导致上层服务业务中断，影响集群的管理维护和业务。	操作前确认操作的必要性，操作时确保同一时间无其它管理维护操作。
	修改OLDAP端口	修改该参数会重启LdapServer和Kerberos服务和其关联的所有服务，影响业务运行。	操作前确认操作的必要性，操作时确保同一时间无其它管理维护操作。
	删除用户绑定的supergroup用户组	删除supergroup用户组会导致相关用户权限变小，影响业务访问。	修改前确认需要添加的权限，确保用户绑定的supergroup权限删除前，相关权限已经添加，不会对业务造成影响。
	重启、停止服务	重启过程中会中断服务，如果勾选同时重启上层服务会导致依赖该服务的上层服务中断。	操作前确认重启的必要性。
	修改节点SSH默认端口	修改默认端口（22）将导致集群健康检查结果中节点互信、omm/ommdba用户密码过期等检查项不准确。	执行相关操作前将SSH端口改回默认值。
ClickHouse	删除ClickHouse数据目录	该操作将会导致业务信息丢失。	请勿手动删除数据目录。
	缩容ClickHouseServer实例	该操作需要关注同分片中的ClickHouseServer实例节点需要同时退服缩容，否则会造成逻辑集群拓扑信息错乱。 该操作执行前需检查逻辑集群内各节点的数据库和数据表信息，进行缩容预分析，保证缩容退服过程中数据迁移成功，避免数据丢失。	进行缩容操作前，提前收集信息进行ClickHouse逻辑集群及实例节点状态判断。观察ClickHouse逻辑集群拓扑信息，各ClickHouseServer中数据库和数据表信息，以及数据量。
	扩容ClickHouseServer实例	该操作需要关注新扩容节点是否需要创建老节点上同名的数据库或数据表，否则会造成后续数据迁移、数据均衡以及缩容退服失败。	进行扩容操作前，确认新扩容ClickHouseServer实例作用和目的，是否需要同步创建相关数据库和数据表。

操作类别	操作名称	操作风险	规避措施
	退服ClickHouseServer实例	该操作需要关注同分片中的ClickHouseServer实例节点需要同时退服，否则会造成逻辑集群拓扑信息错乱。 该操作执行前需检查逻辑集群内各节点的数据库和数据表信息，进行预分析，保证退服过程中数据迁移成功，避免数据丢失。	进行退服操作前，提前收集信息进行Clickhouse逻辑集群及实例节点状态判断。
	入服ClickHouseServer实例	该操作需要关注入服时必须选择原有分片中的所有节点入服，否则会造成逻辑集群拓扑信息错乱。	进行入服操作前，对于待入服节点的分片归属信息需要确认。
	修改数据目录下内容（创建文件、文件夹）	该操作将会导致该节点上的ClickHouse的实例故障。	请勿手动在数据目录下创建或修改文件及文件夹。
	单独启停基础组件	该操作将会影响服务的一些基础功能导致业务失败。	请勿单独启停ZooKeeper/Kerberos/LDAP等基础组件，启停基础组件请勾选关联服务。
	使用内置的default用户以及clickhouse用户	内置的default用户以及clickhouse用户密码具有不确定性，会导致生产业务受损。	新增并使用业务用户。
DBService	修改DBService密码	修改密码需要重启服务，服务在重启过程中无法访问。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	恢复DBService数据	数据恢复后，会丢失从备份时刻到恢复时刻之间的数据。 数据恢复后，依赖DBService的组件可能配置过期，需要重启配置过期的服务。	恢复前确认操作的必要性，恢复时确保同一时间无其它管理维护操作。
	DBService主备倒换	倒换DBServer过程中，DBService无法提供服务。	操作前确认该操作的必要性，操作时确保同一时间无其它管理维护操作。
Flink	修改Flink日志级别	如果修改为DEBUG，会影响任务运行性能。	修改前确认操作的必要性，并及时修改回默认设定。

操作类别	操作名称	操作风险	规避措施
	修改Flink文件权限	该操作可能导致任务运行失败。	修改前确认操作的必要性。
Flume	修改Flume实例的启动参数“GC_OPTS”	导致Flume服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
HBase	修改加密的相关配置项： ● hbase.regionserver.wal.encryption ● hbase.crypto.keyprovider.parameters.uri ● hbase.crypto.keyprovider.parameters.encryptedtext	导致HBase服务启动异常	修改相关配置项时请严格按照提示描述，加密相关配置项是有关联的，确保修改后的值有效
	已使用加密的情况下关闭或者切换加密算法	关闭主要指修改hbase.regionserver.wal.encryption为false，切换主要指AES和SM4（也称为SMS4）的切换。 操作可能导致服务启动失败，数据丢失。	加密HFile和WAL内容的时候，如果已经使用一种加密算法加密并且已经建表，请不要随意关闭或者切换加密算法。 未建加密表（ENCRYPTION=>AES/SMS4）的情况下可以切换，否则禁止操作。
	修改HBase实例的启动参数GC_OPTS、HBASE_HEAPSIZE	导致HBase服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效，且GC_OPTS与HBASE_HEAPSIZE参数值无冲突。
	使用OfflineMetaRepair工具	导致HBase服务启动异常。	必须在HBase下线的情况下才可以使用该命令，而且不能在数据迁移的场景中使用该命令。
HDFS	修改HDFS的NameNode的数据存储目录dfs.namenode.name.dir、DataNode的数据配置目录dfs.datanode.data.dir	导致HDFS服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。

操作类别	操作名称	操作风险	规避措施
	执行**hadoop distcp**命令时使用**-delete**参数	distcp拷贝时，源集群没有而目的集群存在的文件，会在目的集群删除。	在使用Distcp的时候，确保是否保留目的集群多余的文件，谨慎使用**-delete**参数。 Distcp数据拷贝后，查看目的的数据是否按照参数配置保留或删除。
	修改HDFS实例的启动参数GC_OPTS、HADOOP_HEAPSIZE和GC_PROFILE	导致HDFS服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效，且GC_OPTS与HADOOP_HEAPSIZE参数值无冲突。
	修改HDFS的副本数目dfs.replication值由3改为1	• 存储可靠性下降，磁盘故障时，会发生数据丢失。 • NameNode重启失败，HDFS服务不可用。	修改相关配置项时，请仔细查看参数说明。 保证数据存储的副本数不低于2。
	修改Hadoop中各模块的RPC通道的加密方式hadoop.rpc.protection	导致HDFS服务故障及业务异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
Hive	修改Hive实例的启动参数GC_OPTS	修改该参数可能会导致Hive实例无法启动。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	删除MetaStore所有实例	Hive元数据丢失，Hive无法提供服务。	除非确定丢弃Hive所有表信息，否则不要执行该操作。
	使用HDFS文件系统接口或者HBase接口删除或修改Hive表对应的文件	该操作会导致Hive业务数据丢失或被篡改。	除非确定丢弃这些数据，或者确保该修改操作符合业务需求，否则不要执行该操作。
	使用HDFS文件系统接口或者HBase接口修改Hive表对应的文件或目录访问权限	该操作可能会导致相关业务场景不可用。	请勿执行该操作。
	使用HDFS文件系统接口删除或修改文件hdfs:///apps/templeton/hive-3.1.0.tar.gz	该操作可能会导致WebHCat无法正常执行业务。	请勿执行该操作。

操作类别	操作名称	操作风险	规避措施
	导出表数据覆盖写入本地目录，例如将**t1**表中数据导出，覆盖到“/opt/dir”路径下： **insert overwrite local directory '/opt/dir' select * from t1;**	该操作会删除目标目录，如果设置错误，会导致软件或者操作系统无法启动。	确认需要写入的路径下不要包含任何文件，或者不要使用overwrite关键字。
	将不同的数据库、表或分区文件指定至相同路径，例如默认仓库路径“/user/hive/warehouse”。	执行创建操作后数据可能会紊乱，如果删除其中一个数据库、表或分区，会导致其他对象数据丢失。	请勿执行该操作。
IoTDB	删除数据目录	该操作将会导致业务信息丢失。	请勿手动删除数据目录。
	修改数据目录下内容（创建文件、文件夹）	该操作将会导致该节点上的IoTDB的实例故障。	请勿手动在数据目录下创建或修改文件及文件夹。
	单独启停基础组件	该操作将会影响服务的一些基础功能导致业务失败。	请勿单独启停Kerberos/LDAP等基础组件，启停基础组件请勾选关联服务。
Kafka	删除Topic	该操作将会删除已有的主题和数据。	采用Kerberos认证，保证合法用户具有操作权限，并确保主题名称正确。
	删除Kafka数据目录	该操作将会导致业务信息丢失。	请勿手动删除数据目录。
	修改数据目录下内容（创建文件、文件夹）	该操作将会导致该节点上的Broker实例故障。	请勿手动在数据目录下创建或修改文件及文件夹。
	修改磁盘自适应功能“disk.adapter.enable”参数	该操作会在磁盘使用空间达到阈值时调整Topic数据保存周期，超出保存周期的历史数据可能被清除。	若个别Topic不能做保存周期调整，将该Topic配置在“disk.adapter.topic.blacklist”参数中，在KafkaTopic监控页面观察数据的存储周期。
	修改数据目录“log.dirs”配置	该配置不正确将会导致进程故障。	确保所修改或者添加的数据目录为空目录，且权限正确。

操作类别	操作名称	操作风险	规避措施
	减容Kafka集群	该操作将会导致部分Topic数据副本数量减少，可能会导致Topic无法访问。	请先做好数据副本转移工作，然后再进行减容操作。
	单独启停基础组件	该操作将会影响服务的一些基础功能导致业务失败。	请勿单独启停ZooKeeper/Kerberos/LDAP等基础组件，启停基础组件请勾选关联服务。
	删除/修改元数据	修改或者删除ZooKeeper上Kafka的元数据可能导致Topic或者Kafka服务不可用。	请勿删除或者修改Kafka在ZooKeeper上保存的元数据信息。
	修改元数据备份文件	修改Kafka元数据备份文件，并被使用进行Kafka元数据恢复成功后，可能导致Topic或者Kafka服务不可用。	请勿修改Kafka元数据备份文件。
KrbServer	修改KrbServer的参数KADMIN_PORT	修改该参数后，若没有及时重启KrbServer服务和其关联的所有服务，会导致集群内部KrbClient的配置参数异常，影响业务运行。	修改该参数后，请重启KrbServer服务和其关联的所有服务。
	修改KrbServer的参数kdc_ports	修改该参数后，若没有及时重启KrbServer服务和其关联的所有服务，会导致集群内部KrbClient的配置参数异常，影响业务运行。	修改该参数后，请重启KrbServer服务和其关联的所有服务。
	修改KrbServer的参数KPASSWD_PORT	修改该参数后，若没有及时重启KrbServer服务和其关联的所有服务，会导致集群内部KrbClient的配置参数异常，影响业务运行。	修改该参数后，请重启KrbServer服务和其关联的所有服务。
	修改Manager系统域名	若没有及时重启KrbServer服务和其关联的所有服务，会导致集群内部KrbClient的配置参数异常，影响业务运行。	修改该参数后，请重启KrbServer服务和其关联的所有服务。
	配置跨集群互信	该操作会重启KrbServer服务和其关联的所有服务，影响集群的管理维护和业务。	更换前确认操作的必要性，更换时确保同一时间无其它管理维护操作。

操作类别	操作名称	操作风险	规避措施
LdapServer	修改LdapServer的参数LDAP_SERVER_PORT	修改该参数后，若没有及时重启LdapServer服务和其关联的所有服务，会导致集群内部LdapClient的配置参数异常，影响业务运行。	修改该参数后，请重启LdapServer服务和其关联的所有服务。
	恢复LdapServer数据	该操作会重启Manager和整个集群，影响集群的管理维护和业务。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	更换LdapServer所在节点	该操作会导致部署在该节点上的服务中断，且当该节点为管理节点时，更换节点会导致重启OMS各进程，影响集群的管理维护。	更换前确认操作的必要性，更换时确保同一时间无其它管理维护操作。
	修改LdapServer密码	修改密码需要重启LdapServer和Kerberos服务，影响集群的管理维护和业务。	修改前确认操作的必要性，修改时确保同一时间无其它管理维护操作。
	节点重启导致LdapServer数据损坏	如果未停止LdapServer服务，直接重启LdapServer所在节点，可能导致LdapServer数据损坏。	使用LdapServer备份数据进行恢复。
Loader	修改Loader实例的浮动IP地址	导致Loader服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	修改Loader实例的启动参数LOADER_GC_OPTS	导致Loader服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	往HBase导入数据时选择清空表数据	目标表的原数据被清空。	选择时确保目标表的数据可以清空。
Spark	修改配置项“spark.yarn.queue”	导致Spark服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	修改配置项“spark.driver.extraJavaOptions”	导致Spark服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	修改配置项“spark.yarn.cluster.driver.extraJavaOptions”	导致Spark服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。

操作类别	操作名称	操作风险	规避措施
	修改配置项“spark.eventLog.dir”	导致Spark服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	修改配置项“SPARK_DAEMON_JAVA_OPTS”	导致Spark服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	删除所有JobHistory实例	导致历史应用的event log丢失。	至少保留一个JobHistory实例，观察JobHistory中是否可以查看历史应用信息。
	删除或修改HDFS上的spark-archive文件	导致JDBCServer启动异常及业务功能异常。	删除“/user/spark2x/jars/*XXX*/spark-archive-2x.zip”或者“/user/spark/jars/*XXX*/spark-archive.zip”时，等待10~15分钟，zip包自动恢复。
Storm	修改插件相关的配置项： ● storm.scheduler ● nimbus.authorizer ● storm.thrift.transport ● nimbus.blobstore.class ● nimbus.topology.validator ● storm.principal.tolocal	导致Storm服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的类名是存在并有效的。
	修改Storm实例的启动参数GC_OPTS： ● NIMBUS_GC_OPTS ● SUPERVISOR_GC_OPTS ● UI_GC_OPTS ● LOGVIEWER_GC_OPTS	导致Storm服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。

操作类别	操作名称	操作风险	规避措施
	修改用户资源池配置参数“resource.aware.scheduler.user.pools”	导致Storm业务提交后无法正常运行。	修改相关配置项时请严格按照提示描述，确保给每个用户分配的资源合理有效。
	修改Storm数据目录	该操作不当会导致服务异常，无法提供服务。	请勿手动操作数据目录。
	删除/修改Storm元数据	删除Nimbus元数据会导致服务异常，并且已运行业务丢失。	请勿手动删除Nimbus元数据文件。
	修改Storm文件权限	修改元数据目录和日志目录权限不当会引起服务异常。	请勿手动修改文件权限。
	删除Storm拓扑	该操作会删除正在运行中的拓扑。	确保在必要时删除拓扑。
Yarn	删除或者修改数据目录yarn.nodemanager.local-dirs和yarn.nodemanager.log-dirs	该操作将会导致业务信息丢失。	请勿手动删除数据目录。
ZooKeeper	删除或者修改ZooKeeper的数据目录	该操作将会导致业务信息丢失。	修改ZooKeeper目录时候，严格按照扩容指导操作。
	修改ZooKeeper实例的启动参数GC_OPTS	导致ZooKeeper服务启动异常。	修改相关配置项时请严格按照提示描述，确保修改后的值有效。
	设置ZooKeeper中znode的ACL信息	修改ZooKeeper中znode的权限，可能会导致其他用户无权限访问该znode，导致系统功能异常。	修改相关配置项时请确保修改的ACL信息，不会影响其他组件正常使用ZooKeeper。

10 技术支持

MRS服务是租户完全可控的半托管云服务，为用户提供一站式企业级大数据平台，用户可以在MRS集群上轻松运行Hadoop、Hive、Spark、HBase、Kafka、Flink等大数据组件，帮助企业快速构建海量数据信息处理系统，并通过对海量信息数据实时与非实时的分析挖掘，发现全新价值点和企业商机。

维护策略声明

MRS集群资源归属于用户，MRS提供基于该资源的半托管云服务能力，用户拥有对集群的完全控制权，默认情况下，云服务无权限对客户集群进行操作，集群日常运维管理由用户负责，如果在大数据集群运维过程中遇到了相关技术问题，可以联系技术支持团队获得帮助，该技术支持仅协助分析处理MRS云服务相关求助，不包含云服务以外的求助，例如用户基于大数据平台构建的应用系统等。

技术支持范围

- 支持的服务
  - MRS云服务管理控制台提供的相关功能：
    - 集群的创建、删除、扩容、缩容
    - 集群作业管理
    - 集群告警管理
    - 集群补丁管理
    - IAM用户委托管理
    - 对外API接口管理
  - MRS服务提供的开源大数据组件，其中开源组件请参考对应MRS版本组件列表。
  - 支持客户进行MRS服务相关开源组件漏洞分析，如影响分析、修复建议，由用户负责评估对应的业务影响和进行最终实施。
- 不支持的服务
  - 不负责提供具体MRS集群和开源大数据组件管理的运维操作，包括参数配置修改、重启、容量规划、组件性能优化以及集群上任何运维操作等。

- 不负责基于MRS集群之上的客户业务应用开发问题答疑和处理，例如业务设计、代码开发、作业性能调优和业务迁移等。
- 在MRS集群组件服务无明显异常或明确产品质量缺陷的情况下，不负责单个大数据作业运行异常问题的排查分析。
- 不负责在MRS集群上进行非标操作产生的非预期问题分析和解决，如重装操作系统、误删除数据、删除服务目录和文件、修改OS系统配置和文件权限、删除“/etc/hosts”配置、直接后台卸载磁盘、修改节点IP地址、删除创建集群时的默认安全组规则等。
- 不负责对用户在MRS集群环境上自建安装的非MRS提供的第三方组件的问题排查和解决。

11 计费说明

MRS服务计费简单、易于预测。MRS支持按需计费，同时您也可以选择更经济的包年、包月的包周期计费方式。为了便于您便捷的下单购买，在控制台购买界面中已经为您计算好了整个MRS集群的价格，您可一键完成购买。

计费项

购买MRS集群的费用包含两个部分：

- MRS服务管理费用

  说明

  您可以在“费用中心 > 账单管理 > 费用账单”里筛选如下内容查看费用详情。

  图 11-1 查看 MRS 服务管理费用

  - 如果集群版本类型为“LTS版”：按“MRS-LTS服务费用”进行筛选。
  - 如果集群版本类型为“普通版”：
    - 2022年6月及其之前购买的集群：按“MapReduce服务虚拟机”进行筛选。
    - 2022年6月之后购买的集群：按“MRS-BASIC服务费用”进行筛选。

- IaaS基础设施资源费用（弹性云服务器，云硬盘，弹性IP/带宽等）

MRS服务管理费用详情，请参见**产品价格详情**。

您可以通过MRS提供的**价格计算器**，选择您需要的集群节点规格，来快速计算出购买MRS集群的参考价格。

MRS集群删除或退订后不再产生费用。

计费模式

使用MRS的首要操作就是购买MRS集群，MRS当前支持包年包月和按需计费模式。

- 包年/包月：根据集群购买时长，一次性支付集群费用。最短时长为1个月，最长时长为1年。
- 按需计费：节点按实际使用时长计费，计费周期为一小时。

变更配置

在开通MRS前有多种实例供您选择，您可根据业务需要选择合适的Master和Core节点实例。当集群启动后，MRS提供如下几种变更配置的方式。

- 配置Task节点：新增Task节点，请参见**扩容集群**中的“相关任务”。
- 扩容：手动扩容Core或Task节点，请参见**扩容集群**。
- 配置弹性伸缩：根据业务数据量的变化动态调整集群Task节点数量以增减资源，请参见**配置弹性伸缩规则**。

若MRS提供的变更配置方式不满足您的要求，您也可以通过重建集群，然后做数据迁移的方式实现集群配置的变更。

续费

如需续费，请进入“**续费管理**”页面进行续费操作。

欠费

包年/包月集群，没有欠费的概念。

按需购买的集群是按每小时扣费，当余额不足，无法对上一个小时的费用进行扣费，就会导致集群欠费，集群欠费后有**保留期**。您续费后即可解冻集群，可继续正常使用，请注意在保留期进行的续费，是以原到期时间作为生效时间，您应当支付从进入保留期开始到续费时的服务费用。

您购买的集群欠费后，会导致部分操作受限，建议您尽快续费。具体受限操作如下所示：

- 创建集群
- 扩容集群
- 缩容集群
- 新增Task节点
- 升级Master节点规格

服务到期

- 按需购买的集群，没有到期时间。
- 包年/包月集群到期后进入**保留期**，此时无法在MRS管理控制台进行该集群的操作，相关接口也无法调用，自动化监控或告警等运维也会停止。如果在保留期结束时您没有续费，集群将终止服务，系统中的数据也将被永久删除。

保证金

按需购买集群时，华为云根据用户等级和历史使用情况可能会冻结一定的保证金，资源释放时自动解冻保证金。

12 权限管理

如果您需要对华为云上创建的MapReduce服务资源，给企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全地控制华为云资源的访问。

通过IAM，您可以在华为云账号中给员工创建IAM用户，并授权控制用户对资源的访问范围。例如您的员工中有负责应用开发的人员，您希望开发人员拥有MapReduce服务的使用权限，但是不希望其拥有删除MRS集群等高危操作的权限，那么您可以使用IAM为开发人员创建用户，通过授予仅能使用MRS但是不允许删除MRS集群的权限策略，控制用户对MRS集群资源的使用范围。

如果华为云账号已经能满足您的要求，不需要创建独立的IAM用户进行权限管理，您可以跳过本章节，不影响您使用MRS服务的其它功能。

IAM是华为云提供权限管理的基础服务，无需付费即可使用，您只需要为您账号中的资源进行付费。关于IAM的详细介绍，请参见**《IAM产品介绍》**。

MRS 权限说明

默认情况下，管理员创建的IAM用户没有任何权限，需要将其加入用户组，并给用户组授予策略或角色，才能使得用户组中的用户获得对应的权限，这一过程称为授权。授权后，用户就可以基于被授予的权限对云服务进行操作。

MRS部署时通过物理区域划分，为项目级服务。授权时，“作用范围”需要选择“区域级项目”，然后在指定区域对应的项目中设置相关权限，并且该权限仅对此项目生效；如果在“所有项目”中设置权限，则该权限在所有区域项目中都生效。访问MRS时，需要先切换至授权区域。

权限模型根据授权精细程度分为角色和策略。

- 角色：IAM最初提供的一种根据用户的工作职能定义权限的粗粒度授权机制。该机制以服务为粒度，提供有限的服务相关角色用于授权。由于各服务之间存在业务依赖关系，因此给用户授予角色时，可能需要一并授予依赖的其他角色，才能正确完成业务。角色并不能满足用户对精细化授权的要求，无法完全达到企业对权限最小化的安全管控要求。
- 策略：IAM最新提供的一种细粒度授权的能力，可以精确到具体服务的操作、资源以及请求条件等。基于策略的授权是一种更加灵活的授权方式，能够满足企业对权限最小化的安全管控要求。例如：针对MRS服务，管理员能够控制IAM用户仅能对集群进行指定的管理操作。如不允许某用户组删除集群，仅允许操作MRS

集群基本操作，如创建集群、查询集群列表等。多数细粒度策略以API接口为粒度进行权限拆分，MRS支持的API授权项请参见**权限策略和授权项**。

如**表12-1**所示，包括了MRS的所有默认系统策略。

表 12-1 MRS 系统策略

策略名称	描述	策略类别
MRS FullAccess	MRS管理员权限，拥有该权限的用户可以拥有MRS所有权限。	细粒度策略
MRS CommonOperations	MRS服务普通用户权限，拥有该权限的用户可以拥有MRS服务使用权限，无新增、删除资源权限。	细粒度策略
MRS ReadOnlyAccess	MRS服务只读权限，拥有该权限的用户仅能查看MRS的资源。	细粒度策略
MRS Administrator	操作权限： • 对MRS服务的所有执行权限。 • 拥有该权限的用户必须同时拥有Tenant Guest和Server Administrator权限。	RBAC策略

表12-2列出了MRS常用操作与系统权限的授权关系，您可以参照该表选择合适的系统权限。

表 12-2 常用操作与系统策略的授权关系

操作	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
创建集群	√	x	x	√
调整集群	√	x	x	√
升级节点规格	√	x	x	√
删除集群	√	x	x	√
查询集群详情	√	√	√	√
查询集群列表	√	√	√	√
设置弹性伸缩策略	√	x	x	√
查询主机列表	√	√	√	√

操作	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
查询操作日志	√	√	√	√
创建并执行作业	√	√	x	√
停止作业	√	√	x	√
删除单个作业	√	√	x	√
批量删除作业	√	√	x	√
查询作业详情	√	√	√	√
查询作业列表	√	√	√	√
新建文件夹	√	√	x	√
删除文件	√	√	x	√
查询文件列表	√	√	√	√
批量操作集群标签	√	√	x	√
创建单个集群标签	√	√	x	√
删除单个集群标签	√	√	x	√
按照标签查询资源列表	√	√	√	√
查询集群标签	√	√	√	√
访问Manager页面	√	√	x	√
查询补丁列表	√	√	√	√
安装补丁	√	√	x	√
卸载补丁	√	√	x	√

操作	MRS FullAccess	MRS CommonOperations	MRS ReadOnlyAccess	MRS Administrator
运维通道授权	√	√	x	√
运维通道日志共享	√	√	x	√
查询告警列表	√	√	√	√
订阅告警消息提醒	√	√	x	√
提交SQL语句	√	√	x	√
查询SQL结果	√	√	x	√
取消SQL执行任务	√	√	x	√

MRS FullAccess 策略内容

```
{
   "Version": "1.1",
   "Statement": [
      {
         "Action": [
            "mrs:*:*",
            "ecs:*:*",
            "bms:*:*",
            "evs:*:*",
            "vpc:*:*",
            "kms:*:*",
            "rds:*:*",
            "bss:*:*"
         ],
         "Effect": "Allow"
      }
   ]
}
```

MRS CommonOperations 策略内容

```
{
   "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",
            "kms:*:get*",
            "kms:*:list*",
            "rds:*:get*",
            "rds:*:list*",
            "mrs:bootstrap:*",
            "bss:*:view*"
         ],
         "Effect": "Allow"
      },
      {
         "Action": [
            "mrs:cluster:create",
            "mrs:cluster:resize",
            "mrs:cluster:scaleUp",
            "mrs:cluster:delete",
            "mrs:cluster:policy"
         ],
         "Effect": "Deny"
      }
   ]
}
```

MRS ReadOnlyAccess 策略内容

```
{
   "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*",
```

```
            "kms:*:get*",
            "kms:*:list*",
            "rds:*:get*",
            "rds:*:list*",
            "bss:*:view*"
        ],
        "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 策略内容

```
{
  "Depends": [
    {
      "catalog": "BASE",
      "display_name": "Server Administrator"
    },
    {
      "catalog": "BASE",
      "display_name": "Tenant Guest"
    }
  ],
  "Version": "1.0",
  "Statement": [
    {
      "Action": [
        "MRS:MRS:*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

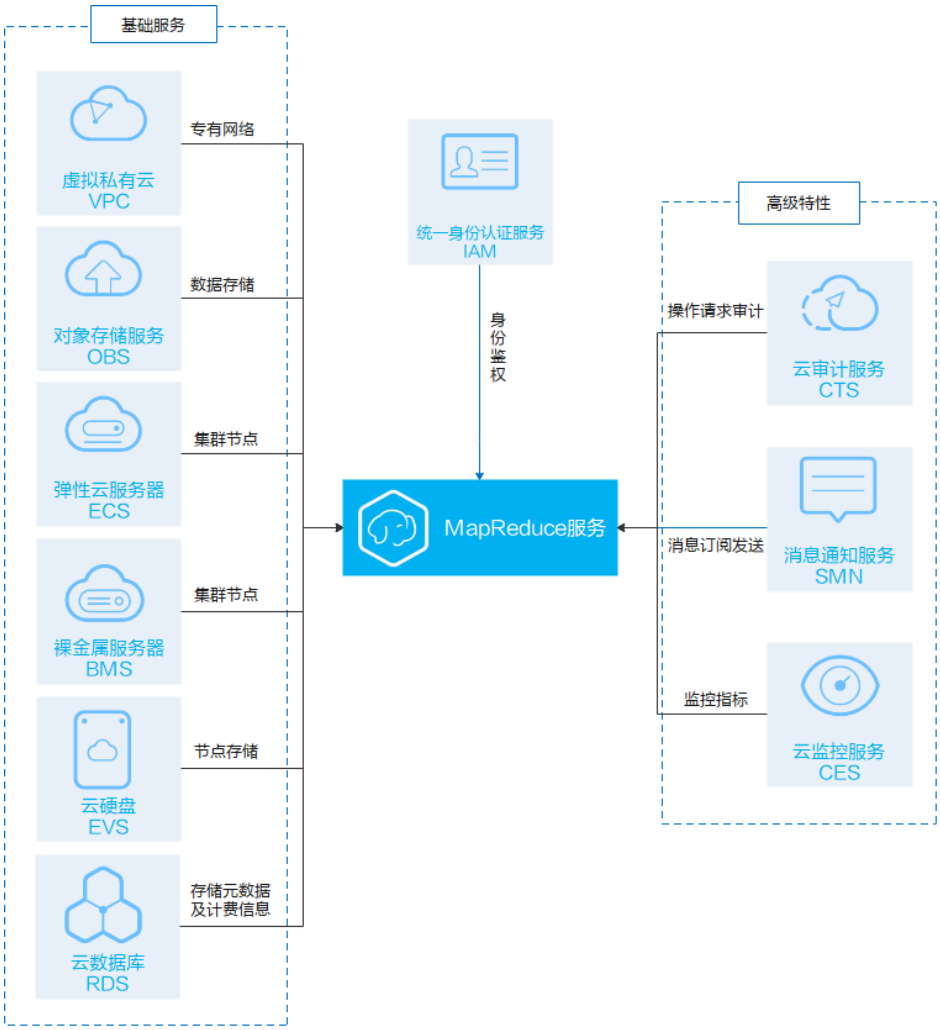
相关链接

- IAM产品介绍
- 创建用户组、用户并授予MRS权限
- 策略支持的授权项

13 与其他云服务的关系

MRS服务与周边其他云服务的关系如图13-1所示。

图 13-1 MRS 与其他云服务的关系图



MRS 服务与其他云服务的关系

表 13-1 MRS 服务与其他云服务的关系

服务名称	MRS服务与其他服务的关系	主要交互功能
虚拟私有云（Virtual Private Cloud）	MRS集群创建在虚拟私有云（VPC）的子网内，VPC通过逻辑方式进行网络隔离，为用户的MRS集群提供安全、隔离的网络环境。	创建虚拟私有云和子网
对象存储服务（Object Storage Service）	对象存储服务（OBS）用于存储用户数据，包括MRS作业输入数据和作业输出数据： • MRS作业输入数据：用户程序和数据文件 • MRS作业输出数据：作业输出的结果文件和日志文件 MRS中HDFS、Hive、MapReduce、Yarn、Spark、Flume和Loader等组件支持从OBS导入、导出数据。 MRS使用OBS的并行文件系统提供服务。	配置存算分离集群（委托方式） 配置存算分离集群（AKSK方式）
弹性云服务器（Elastic Cloud Server）	MRS服务使用弹性云服务器（ECS）作为集群的节点，每个弹性云服务器是集群中的一个节点。	准备运行环境 创建集群
云数据库（Relational Database Service）	云数据库（RDS）用于存储MRS系统运行数据，包括MRS集群元数据和用户计费信息等。	配置数据连接
统一身份认证服务（Identity and Access Management）	统一身份认证服务（IAM）为MRS提供了鉴权功能。	创建用户并授权使用MRS MRS自定义策略 IAM用户同步MRS
消息通知服务（Simple Message Notification）	MRS联合消息通知服务（SMN），采用主题订阅模型，提供一对多的消息订阅以及通知功能，能够实现一站式集成多种推送通知方式。	配置作业消息通知
云审计服务（Cloud Trace Service）	云审计服务（CTS）为用户提供MRS资源操作请求及请求结果的操作记录，供用户查询、审计和回溯使用。	云审计支持的MRS操作列表
云硬盘（Elastic Volume Service）	云硬盘可以为MRS使用的弹性云服务器提供高可靠、高性能、规格丰富并且可弹性扩展的块存储服务，可满足不同场景的业务需求。	扩容数据盘
云监控服务（Cloud Eye）	云监控服务为用户提供立体化监控平台。使您全面了解MRS的资源使用情况、业务的运行状况，并及时收到异常告警做出反应，保证业务顺畅运行。	-

服务名称	MRS服务与其他服务的关系	主要交互功能
裸金属服务器（Bare Metal Server）	裸金属服务为MRS提供专属的云上物理服务器，提供优秀的计算性能以及数据安全。	MRS所使用的裸金属服务器规格

表 13-2 云审计支持的 MRS 操作列表

操作名称	资源类型	事件名称
创建集群	cluster_mrs	createCluster
删除集群	cluster_mrs	deleteCluster
集群扩容	cluster_mrs	scaleOutCluster
集群缩容	cluster_mrs	scaleInCluster
批量终止作业	job_mrs	batch_delete_job
终止作业	job_mrs	kill_job
提交作业	job_mrs	submit_job
集群节点授权	cluster_mrs	authorizedOM

在您开启了云审计服务后，系统开始记录云服务资源的操作。云审计服务管理控制台保存最近7天的操作记录。相关操作步骤请参考“云审计服务（CTS）> 用户指南”。

14 配额说明

配额是用户账号在对应环境配置的可用资源额度，限定配额仅是为了防止资源滥用。

MapReduce服务通常使用的基础资源如下，配额由各个基础服务管理，如需扩大配额，请联系对应服务的技术支持进行扩容：

- 弹性云服务器
- 裸金属服务器
- 虚拟私有云
- 云硬盘
- 镜像服务
- 对象存储服务
- 弹性公网IP
- 消息通知服务
- 统一身份认证服务

其配额查看及修改请参见**关于配额**。

15 常见概念

HBase 表

HBase的表是三个维度排序的映射。从行主键、列主键和时间戳映射为单元格的值。所有的数据存储在HBase的表单元格中。

列

HBase表的一个维度。列名称的格式为“<family>:<label>”，<family>和<label>为任意字符组合。表由<family>的集合组成（<family>又称为列族）。HBase表中的每个列都归属于某个列族。

列族

列族是预定义的列集合，存储在HBase Schema中。如果需要在列族下创建一些列，首先需创建列族。列族将HBase中具有相同性质的数据进行重组，且没有类型的限制。同一列族的每行数据存储在同一个服务器中。每个列族像一个属性，如压缩包、时间戳、数据块缓存等。

MemStore

MemStore是HBase存储的核心，当WAL中数据存储达到一定量时，加载到MemStore进行排序存储。

RegionServer

RegionServer是HBase集群运行在每个工作节点上的服务。一方面维护Region的状态，提供对于Region的管理和服务；另一方面，上传Region的负载信息，参与Master的分布式协调管理。

时间戳

用于索引同一份数据的不同版本，时间戳的类型是64位整型。时间戳可以由HBase在数据写入时自动赋值或者由客户显式赋值。

Store

HBase存储的核心，一个Store拥有一个MemStore和多个StoreFile，一个Store对应一个分区中表的列族。

索引

一种数据结构，提高了对数据库表中的数据检索效率。可以使用一个数据库表中的一列或多列，提供了快速随机查找和有效访问有序记录的基础。

协处理器

HBase提供的在RegionServer执行的计算逻辑的接口。协处理器分两种类型，系统协处理器可以全局导入RegionServer上的所有数据表，表协处理器即是用户可以指定一张表使用协处理器。

Block Pool

Block Pool是隶属于单个Namespace的块的集合。DataNode存储来自集群中所有块池的块。每个块池都是独立管理的。这就允许一个Namespace为新块生成块ID，而不需要和其他Namespace合作。一个NameNode失效，不会影响DataNode为集群中其他NameNode提供服务。

DataNode

HDFS集群的工作节点。根据客户端或者是元数据节点的调度存储和检索数据，定期向元数据及客户端发送所存储的文件块的列表。

文件块

HDFS中存储的最小逻辑单元。每个HDFS文件由一个或多个文件块存储。所有的文件块存储在DataNode中。

文件块副本

一个副本是存储在HDFS中的一些文件块拷贝件。同一个文件块存储多个拷贝件主要用于系统的可用性和容错。

NodeManager

负责执行应用程序的容器，同时监控应用程序的资源使用情况（CPU、内存、硬盘、网络）并且向ResourceManager汇报。

ResourceManager

集群的资源管理器，基于应用程序对资源的需求进行调度。资源管理器提供一个调度策略的插件，它负责将集群资源分配给多个队列和应用程序。调度插件可以基于现有的能力调度和公平调度模型。

Kafka 分区

每一个Topic可以被分为多个Partition，每个Partition对应一个可持续追加的有序且不可变的log文件。

跟随者

跟随者（Follower）负责处理读请求的模块，配合Leader一起进行写请求处理。也可作为Leader的储备，当Leader故障时从Follower当中选举出Leader，避免出现单点故障。

观察者

观察者（Observer）不参与选举和写请求的投票，只负责处理读请求、并向Leader转发写请求，避免系统处理能力浪费。

离散流

Spark Streaming提供的抽象概念。表示一个连续的数据流，是从数据源获取或者通过输入流转换生成的数据流。从本质上说，一个DStream表示一系列连续的RDD。

堆内存（Heap Memory）

堆是JVM运行时数据区域，所有类实例和数组的内存均从此处分配。初始堆内存根据JVM启动参数-Xms控制。

- 最大堆内存（Maximum Heap Memory）：系统可以分配给程序的最大堆内存，JVM启动参数-Xmx指定。
- 分配的堆内存（Committed Heap Memory）：为保证程序运行系统分配的堆内存总量，在程序运行期间根据使用情况，会在初始堆内存和最大堆内存之间波动变化。
- 使用的堆内存（Used Heap Memory）：当前程序运行时已经使用的堆内存，这个内存小于分配的堆内存。
- 非堆内存：在JVM中堆之外的内存称为非堆内存（Non Heap Memory），JVM自身运行时所需要的内存区域，非堆内存有多个内存池，通常包括以下3个部分：
  - 代码缓存区（Code Cache）：主要用于存放JIT所编译的代码。默认限制240MB，可以通过JVM启动参数-XX:InitialCodeCacheSize -XX:ReservedCodeCacheSize进行设置。
  - 类指针压缩空间（Compressed Class Space）：存储类指针的元数据，默认限制1024MB，通过JVM启动参数-XX:CompressedClassSpaceSize进行设置。
  - 元空间（Metaspace）：用于存放元数据，通过JVM启动参数-XX:MetaspaceSize -XX:MaxMetaspaceSize进行设置。
- 最大非堆内存（Maximum Non Heap Memory）：系统可以分配给程序的最大非堆内存。其值为代码缓存区、类指针压缩空间、元空间最大值之和。
- 分配的非堆内存（Committed Non Heap Memory）：为保证程序运行的系统非堆内存总量，在程序运行期间根据使用的非堆内存情况，会在初始非堆内存和最大非堆内存之间波动变化。
- 使用非堆内存（Used Non Heap Memory）：当前程序运行时已经使用的非堆内存，这个值小于分配的非堆内存。

Hadoop

一个分布式系统框架。用户可以在不了解分布式底层细节的情况下，开发分布式程序，充分利用了集群的高速运算和存储。Hadoop能够对大量数据以可靠的、高效的、可伸缩的方式进行分布式处理。Hadoop是可靠的，因为它假设计算单元和存储会失

败，因此维护多个工作数据副本，确保对失败节点重新分布处理；Hadoop是高效的，因为它以并行的方式工作，从而加快处理速度；Hadoop是可伸缩的，能够处理PB级数据。Hadoop主要由HDFS、MapReduce、HBase和Hive等组成。

角色

角色是服务的组成要素，每个服务由一个或多个角色组成。服务通过角色安装到主机（即服务器）上，保证服务正常运行。

集群

将多个服务器集中起来使它们能够像一台服务器一样提供服务的计算机技术。采用集群通常是为了提高系统的稳定性、可靠性、数据处理能力或服务能力。例如，可以减少单点故障、共享存储资源、负荷分担或提高系统性能等。

实例

当一个服务的角色安装到主机上，即形成一个实例。每个服务有各自对应的角色实例。

元数据（Metadata）

元数据又称中介数据、中继数据，为描述数据的数据，主要是描述数据属性的信息，用来支持如指示存储位置、历史数据、资源查找、文件记录等功能。