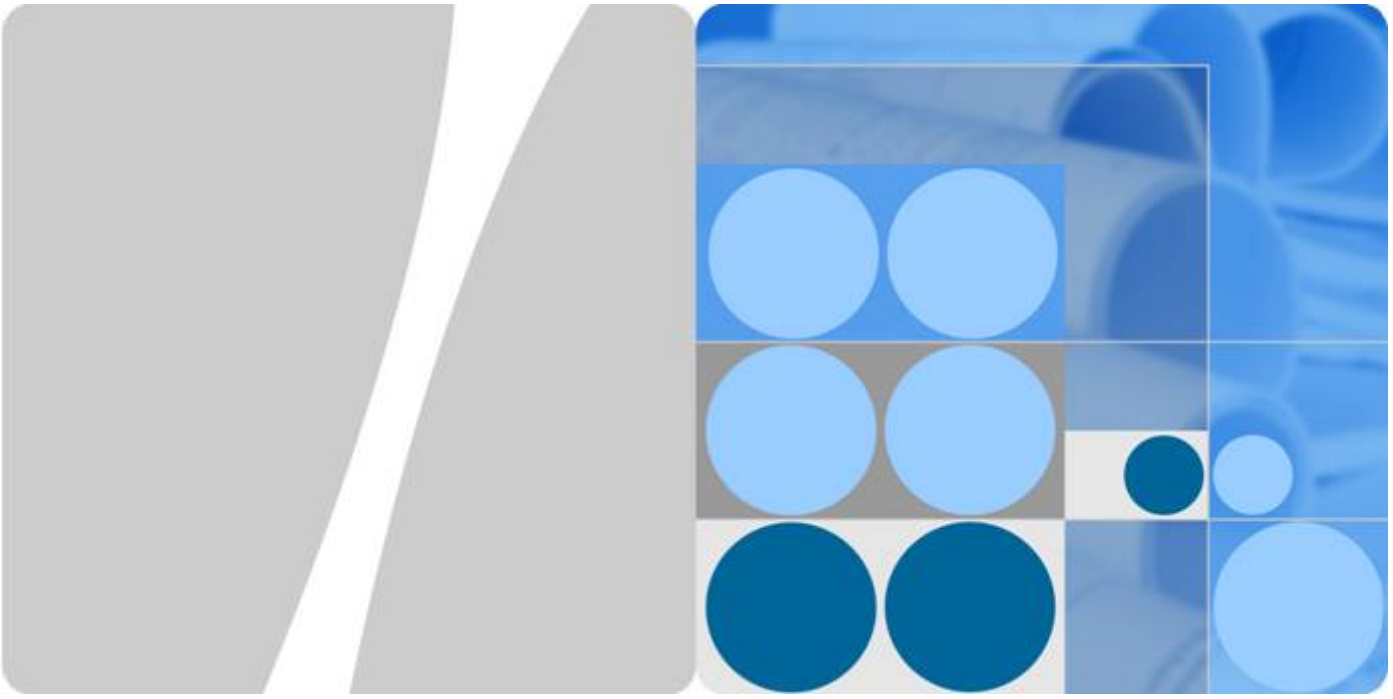


FusionInsight HD

V100R002C60SPC200

产品描述

文档版本　05

发布日期　2017-10-30

华为技术有限公司

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，华为公司对本文档内容不做任何明示或默示的声明或保证。

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。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://e.huawei.com

前 言

概述

本文档简要介绍了FusionInsight HD的产品介绍、系统架构介绍、部署方案、企业级增强特性介绍等信息。

读者对象

本文档专供需要了解和使用FusionInsight HD软件的用户使用。

本文档的目标读者为：

- 规划工程师
- 运维工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	以本标志开始的文本表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。
警告	以本标志开始的文本表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
注意	以本标志开始的文本表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	以本标志开始的文本能帮助您解决某个问题或节省您的时间。
说明	以本标志开始的文本是正文的附加信息，是对正文的强调和补充。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 05 (2017-10-30)

第五次发布。

文档版本 04 (2017-07-25)

第四次发布。

文档版本 03 (2017-01-20)

第三次发布。

文档版本 02 (2016-09-30)

第二次发布。

文档版本 01 (2016-05-13)

第一次发布。

目 录

1 产品介绍

关于本章

1.1 产品定位

产品简介

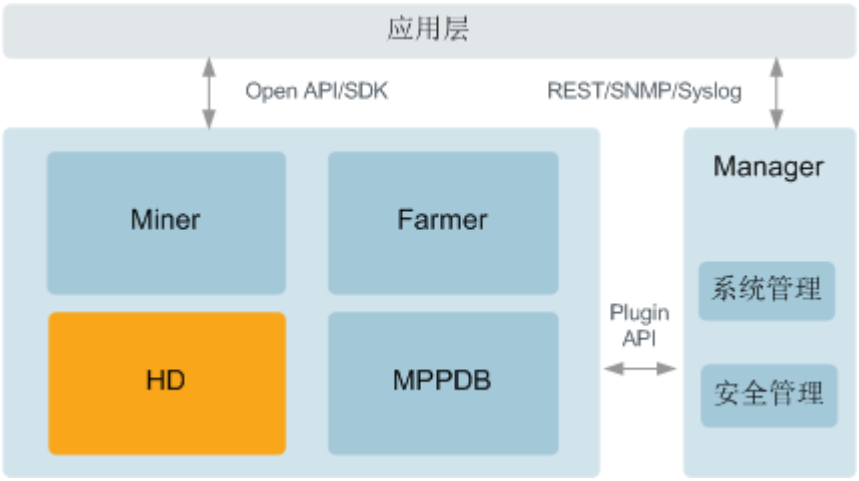
华为FusionInsight HD是一个分布式数据处理系统，对外提供大容量的数据存储、查询和分析能力，可解决各大企业的以下需求：

- 快速地整合和管理不同类型的大容量数据
- 对原生形式的信息提供高级分析
- 可视化所有的可用数据，供特殊分析使用
- 为构建新的分析应用程序提供了开发环境
- 工作负荷的优化和调度

在 FusionInsight 解决方案中的位置

FusionInsight HD在FusionInsight解决方案中的位置如**图1-1**所示。

图 1-1 FusionInsight HD 的位置



FusionInsight是华为企业级大数据存储、查询、分析的统一平台，能够帮助企业快速构建海量数据信息处理系统，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。

FusionInsight解决方案由4个子产品FusionInsight HD、FusionInsight MPPDB、FusionInsight Miner、FusionInsight Farmer和1个操作运维系统FusionInsight Manager构成。

- FusionInsight HD：企业级的大数据处理环境，是一个分布式数据处理系统，对外提供大容量的数据存储、分析查询和实时流式数据处理分析能力。
- FusionInsight MPPDB：企业级的大规模并行处理关系型数据库。FusionInsight MPPDB采用MPP(Massive Parallel Processing)架构，支持行存储和列存储，提供PB(Petabyte，2的50次方字节)级别数据量的处理能力。
- FusionInsight Miner：企业级的数据分析平台，基于华为FusionInsight HD的分布式存储和并行计算技术，提供从海量数据中挖掘出价值信息的平台。
- FusionInsight Farmer：企业级的大数据应用容器，为企业业务提供统一开发、运行和管理的平台。
- FusionInsight Manager：企业级大数据的操作运维系统，提供高可靠、安全、容错、易用的集群管理能力，支持大规模集群的安装部署、监控、告警、用户管理、权限管理、审计、服务管理、健康检查、问题定位、升级和补丁等功能。

1.2 技术特点

华为FusionInsight HD发行版紧随开源社区的最新技术，快速集成最新组件，并在可靠性、安全性、管理性等方面做企业级的增强，持续改进，持续保持技术领先。

FusionInsight HD的企业级增强主要表现在以下几个方面。

安全

- 架构安全

  FusionInsight HD基于开源组件实现功能增强，保持100%的开放性，不使用私有架构和组件。
- 认证安全

- 基于用户和角色的认证统一体系，遵从帐户/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，FusionInsight HD使用LDAP作为帐户管理系统，并通过Kerberos对帐户信息进行安全认证。
- 提供单点登录，统一了Manager系统用户和组件用户的管理及认证。
- 对登录FusionInsight Manager的用户进行审计。

- 文件系统层加密

  Hive、HBase可以对表、字段加密，集群内部用户信息禁止明文存储。
  - 加密灵活：加密算法插件化，可进行扩充，亦可自行开发。非敏感数据可不加密，不影响性能（加密约有5%性能开销）。
  - 业务透明：上层业务只需指定敏感数据（Hive表级、HBase列族级加密），加解密过程业务完全不感知。

可信

- 所有管理节点组件均实现HA（High Availability）

  业界第一个实现所有组件HA的产品，确保数据的可靠性、一致性。NameNode、Hive Server、HMaster、Resources Manager等管理节点均实现HA。
- 集群异地灾备

  业界第一个支持超过1000公里异地容灾的大数据平台，为日志详单类存储提供了迄今为止可靠性最佳实践。
- 数据备份恢复

  表级别集群全量备份、增量备份，数据恢复（对本地存储的业务数据进行完整性校验，在发现数据遭破坏或丢失时进行自恢复）。

易用

- 统一运维管理

  Manager作为FusionInsight HD的运维管理系统，提供界面化的统一安装、告警、监控和集群管理。
- 易集成

  提供北向接口，实现与企业现有网管系统集成；当前支持Syslog接口，接口消息可通过配置适配现有系统；整个集群采用统一的集中管理，未来北向接口可根据需求灵活扩展。
- 易开发

  提供自动化的二次开发助手和开发样例，帮助软件开发人员快速上手。

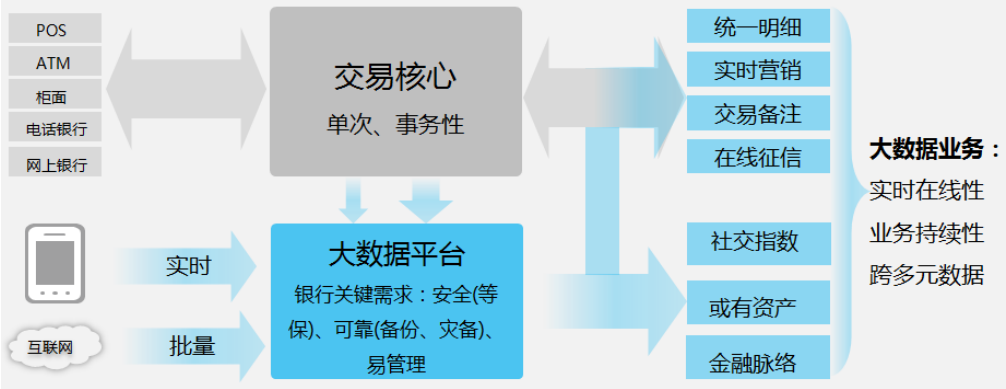
1.3 应用场景

1.3.1 金融领域

应用场景

金融领域典型应用场景如图1-2所示。

图 1-2 金融领域应用示意图



场景特点

面对互联网金融的竞争压力，金融企业急需重构以大数据分析挖掘为基础的决策和服务体系，提升自身竞争力和客户满意度。在大数据时代，银行将从以交易为中心转向以数据为中心，以应对更多维、更大量、更实时的数据和互联网业务的挑战。

华为FusionInsight HD可以从不同方面解决金融企业的问题，提升其竞争力。例如：

- 历史交易明细实时查询业务

  实时查询用户的历史交易明细，能够将查询范围从1年提升到7年以上；能够实现百TB级历史数据表的毫秒级查询。
- 实时征信业务

  用户信用卡征信时间由3天左右减少到10分钟以内。
- 小微贷业务预测

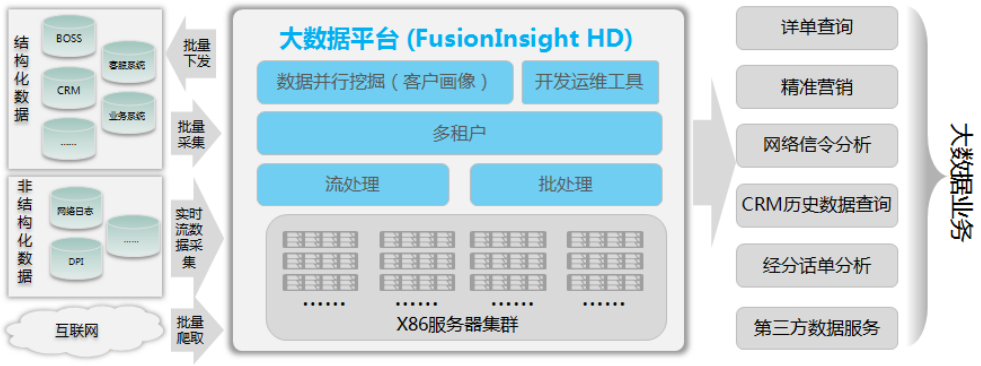
  TOP 1000小微贷倾向用户预测准确率比传统模式提高40倍以上。
- 精准营销
  - 大大缩短分布式网银日志的收集周期，基于网银日志的用户行为统计与分析，提供精确营销，极大提升了网银用户体验效果。
  - 只需不到原来20%的推荐短信，就可基本覆盖原来全部的有效购买用户，实现精准营销。

1.3.2 运营商领域

应用场景

运营商领域的应用场景示意图如图1-3所示。

图 1-3 运营商领域应用示意图



场景特点

随着大数据时代的到来，运营商面临如下挑战。

- 需要处理的数据数量、种类呈现爆炸式增长，尤其对于非结构化数据的处理，现有架构的分析速度十分缓慢。
- 现有应用系统以烟囱式建设，导致数据重复存储，跨系统数据共享难度大，业务决策分析缓慢。

华为FusionInsight HD产品可以从不同方面解决运营商问题，提升其竞争力。例如：

- 构建统一的大数据详单集中平台和经营详单数据分析平台，从架构上根本解决运营商问题。
  - 历史话单查询，客户可实时查询的历史话单由3个月提升到6个月至24个月。
  - 经营详单数据并发分析，由原来的5天减少到1天。
- 构建统一的PB级大数据平台，统一存储业务数据。利用大数据平台分布式计算能力，并发处理各种分析任务，快速获取业务决策结果。
  - 缩短新业务推出周期，由原来的1.5个月减少到1周。
  - 存量用户挽留，VIP用户离网率大幅降低。
- 在保障数据安全性和隐私性前提下，提供数据共享访问和开放接口，让大数据对外提供共享服务，支撑业务创新与商业成功。

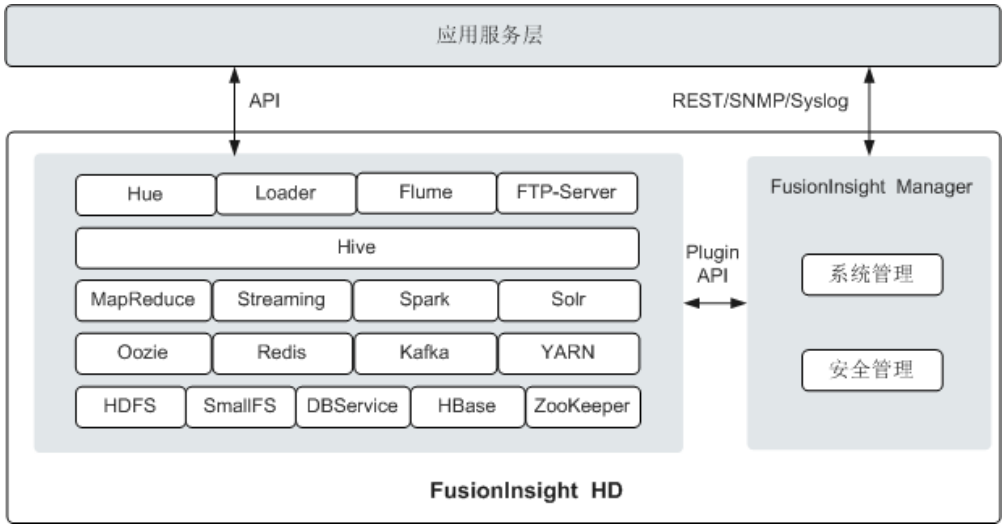
2 系统架构

关于本章

2.1 软件组成

FusionInsight HD系统的逻辑架构图如图2-1所示。

图 2-1 FusionInsight HD 系统逻辑架构图



FusionInsight HD对开源组件进行封装和增强，包含Manager和众多组件，分别提供功能如下：

- Manager

作为运维系统，为FusionInsight HD提供高可靠、安全、容错、易用的集群管理能力，支持大规模集群的安装部署、监控、告警、用户管理、权限管理、审计、服务管理、健康检查、问题定位、升级和补丁等。

- Hue

  提供了FusionInsight HD应用的图形化用户Web界面。Hue支持展示多种组件，目前支持HDFS、YARN、Hive和Solr。

- Loader

  实现FusionInsight HD与关系型数据库、文件系统之间交换数据和文件的数据加载工具；同时提供REST API接口，供第三方调度平台调用。

- Flume

  一个分布式、可靠和高可用的海量日志聚合系统，支持在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，并写入各种数据接受方（可定制）的能力。

- FTP-Server

  通过通用的FTP客户端、传输协议提供对HDFS文件系统进行基本的操作，例如：文件上传、文件下载、目录查看、目录创建、目录删除、文件权限修改等。

- 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。

- 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。

- Streaming

  提供分布式、高性能、高可靠、容错的实时计算平台，可以为海量数据提供实时处理。CQL（Continuous Query Language）提供的类SQL流处理语言，可以快速进行业务开发，缩短业务上线时间。

- Spark

  基于内存进行计算的分布式计算框架。

- Solr

  一个高性能，基于Lucene的全文检索服务器。Solr对Lucene进行了扩展，提供了比Lucene更为丰富的查询语言，同时实现了可配置、可扩展，并对查询性能进行了优化，并且提供了一个完善的功能管理界面，是一款非常优秀的全文检索引擎。

- Oozie

  提供了对开源Hadoop组件的任务编排、执行的功能。以Java Web应用程序的形式运行在Java servlet容器（如：Tomcat）中，并使用数据库来存储工作流定义、当前运行的工作流实例（含实例的状态和变量）。

- Redis

  一个开源的、高性能的key-value分布式存储数据库，支持丰富的数据类型，弥补了memcached这类key-value存储的不足，满足实时的高并发需求。

- Kafka

  一个分布式的、分区的、多副本的实时消息发布和订阅系统。提供可扩展、高吞吐、低延迟、高可靠的消息分发服务。

- 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。

- 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。

- SmallFS

  提供小文件后台合并功能，能够自动发现系统中的小文件(通过文件大小阈值判断)，在闲时进行合并，并把元数据存储到本地的LevelDB中，来降低NameNode压力，同时提供新的FileSystem接口，让用户能够透明的对这些小文件进行访问。

- DBService

  一个具备高可靠性的传统关系型数据库，为Hive、Hue、Spark组件提供元数据存储服务。

- HBase

  提供海量数据存储功能，是一种构建在HDFS之上的分布式、面向列的存储系统。

- 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。

2.2 对外接口

FusionInsight HD各组件提供的对外接口说明如表2-1所示。

表 2-1 组件对外接口

组件名	支持的接口类型
HBase	Command-line interface (CLI)、Java、Sqlline、JDBC、REST、Thrift
HDFS	CLI、Java、C、HTTP REST
MapReduce	Java
Hive	JDBC、Thrift、Python、REST、ODBC
Spark	Java、Scala、Python、JDBC
Solr	Java
Oozie	CLI、Java、REST
YARN	CLI、Java、REST
Flume	Java
Loader	REST、CLI
ZooKeeper	Java、CLI
Kafka	Java、Scala
Manager	REST、SNMP、Syslog
Streaming	REST、Java、Thrift

组件名	支持的接口类型
Redis	REST、Java

2.3 组件介绍

2.3.1 Manager

2.3.1.1 基本原理

功能

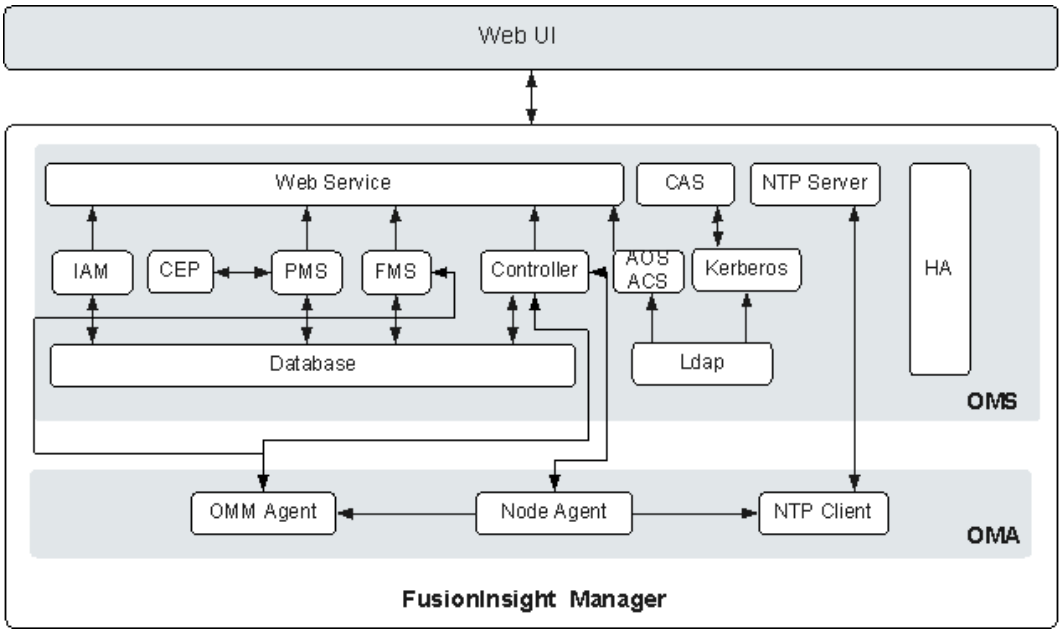
FusionInsight Manager是FusionInsight HD的运维管理系统，为部署在集群内的服务提供统一的集群管理能力。

Manager支持大规模集群的安装部署、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、升级和补丁等功能。

结构

Manager的整体逻辑架构如图2-2所示。

图 2-2 Manager 逻辑架构



FusionInsight 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2-2中各模块的说明如表2-2所示：

表 2-2 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，统一向管理员展示，以及负责接收来自管理员的操作指令，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 Server 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NodeAgent。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。
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，搜集每一个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。

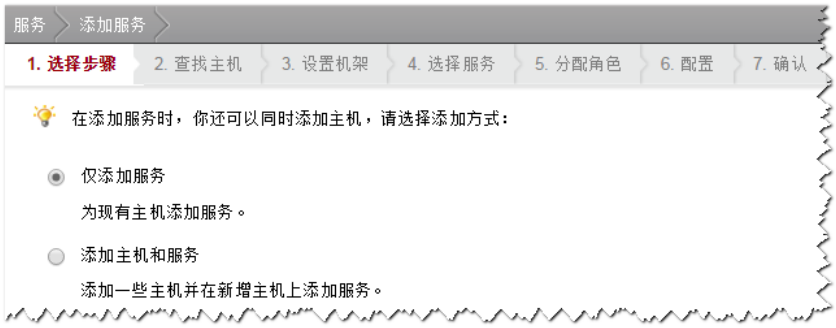
模块名称	描述
Kerberos	在OMS与集群中各部署一个。 ● OMS Kerberos提供单点登录及Controller与Node Agent间认证的功能。 ● 集群中Kerberos提供组件用户安全认证功能，其服务名称为KrbServer，包含两种角色实例： – KerberosServer：认证服务器，为FusionInsight HD提供安全认证使用。 – KerberosAdmin：管理Kerberos用户的进程。
Ldap	在OMS与集群中各部署一个。 ● OMS Ldap在集群安装前为用户认证提供数据存储，在集群安装后作为集群中Ldap的备份。 ● 集群中Ldap为用户认证提供数据存储，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。

2.3.1.2 关键特性

一键式安装

Manager的WebUI提供向导式的集群安装步骤，根据界面用户可一步步完成集群安装。

图 2-3 集群一键式安装



统一监控告警

Manager提供可视化、便捷的监控告警功能。通过“系统概览”，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实时上报告警，界面帮助提供性能指标和告警恢复的详细方法，帮助用户快速解决故障。

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。

单点登录

提供Manager WebUI与组件WebUI之间的单点登录，以及FusionInsight HD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。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，页面将自动锁定。

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。

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FusionInsight HD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。

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。

2.3.2 KrbServer 及 LdapServer

2.3.2.1 基本原理

简介

为了管理FusionInsight集群中数据与资源的访问控制权限，华为大数据平台推荐以安全模式安装集群。在安全模式下，客户端应用程序在访问FusionInsight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。FusionInsight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。

结构

FusionInsight用户登录时安全认证功能主要依赖于Kerberos和LDAP。

图 2-4 安全认证场景架构

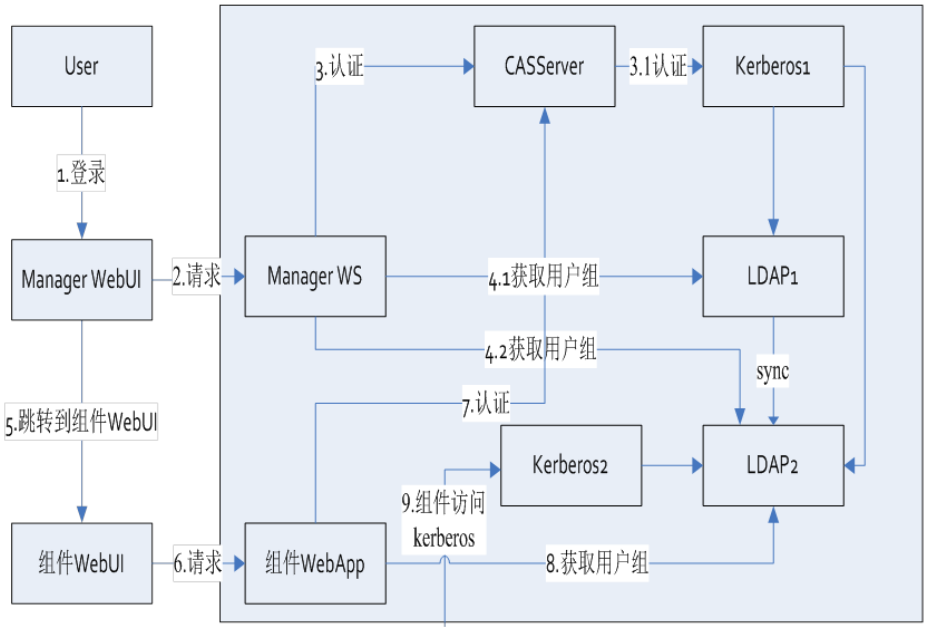


图2-4可分为三类场景：

- 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 2-3 关键模块解释

名称	含义
Manager	FusionInsight Manager
Manager WS	FusionInsight 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（管理平面）服务

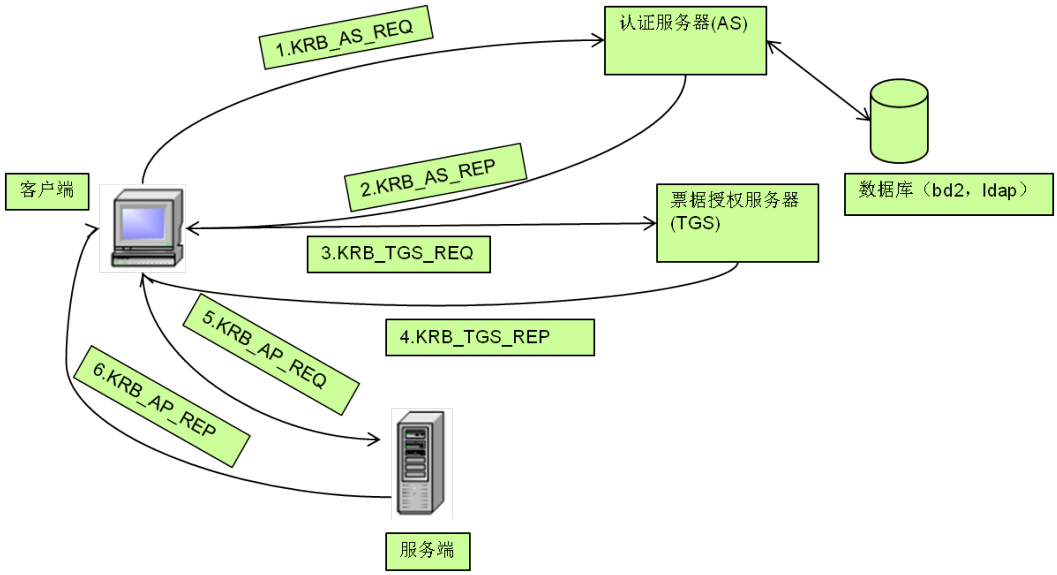
Kerberos1访问LDAP数据：以负载均衡方试访问主备LDAP1两个实例和主备LDAP2两个实例。只能在主LDAP2主实例上进行数据的写操作，可以在LDAP1或者LDAP2上进行数据的读操作。

Kerberos2访问LDAP数据：只能访问主备LDAP2两个实例，且只能在主LDAP2实例进行数据的写操作。

原理

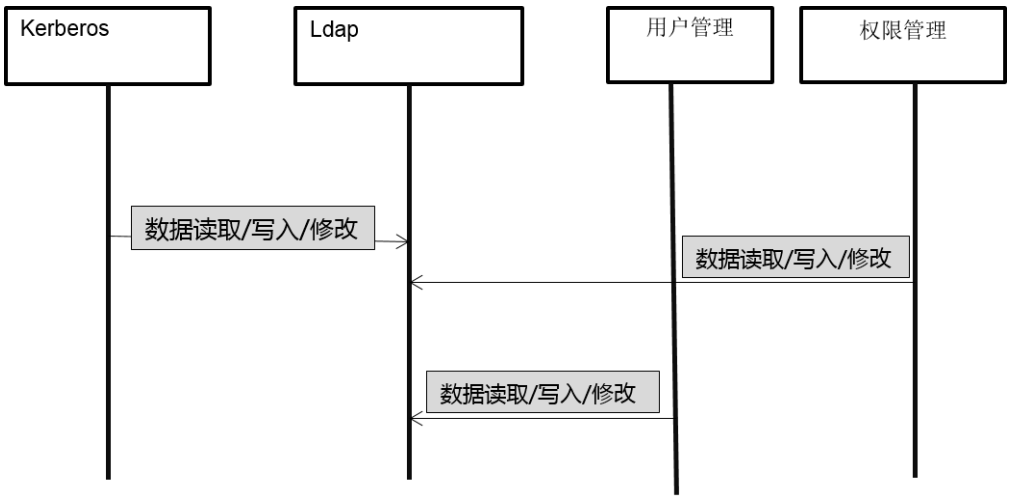
Kerberos认证

图 2-5 认证流程图



LDAP数据读写

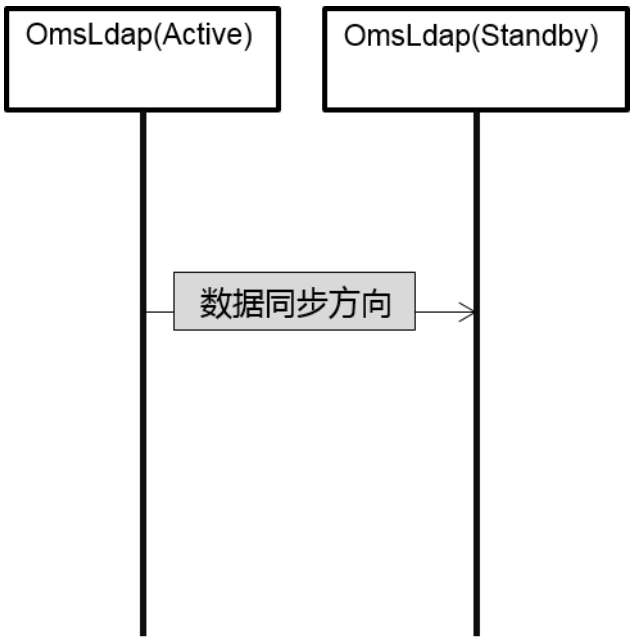
图 2-6 数据修改过程



LDAP数据同步

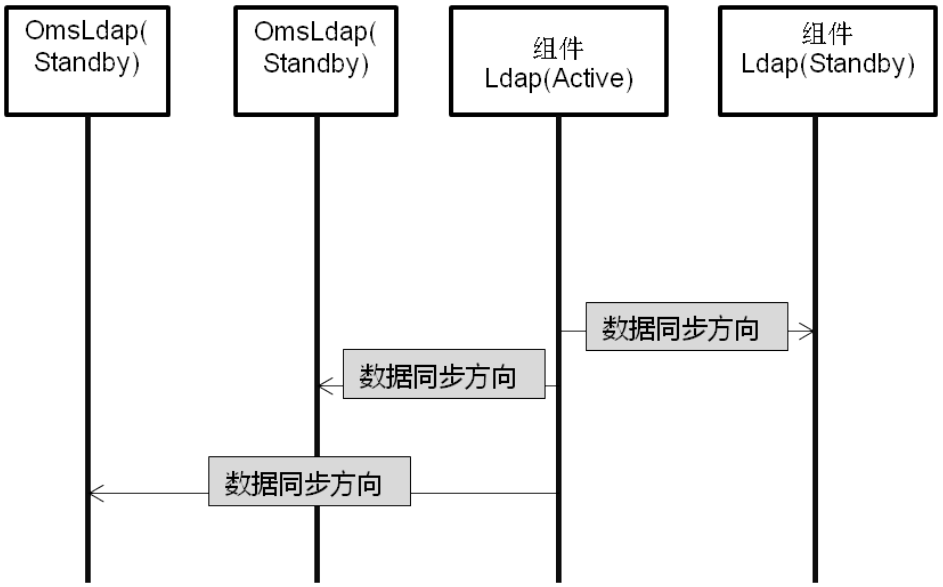
- 安装集群前OMS LDAP数据同步

  图 2-7 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 2-8 LDAP 数据同步



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

2.3.3 HBase

2.3.3.1 基本原理

简介

HBase是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统。HBase适合于存储大表数据（表的规模可以达到数十亿行以及数百万列），并且对大表数据的读、写访问可以达到实时级别。

- 利用Hadoop HDFS（Hadoop Distributed File System）作为其文件存储系统，提供高可靠性、高性能、列存储、可伸缩、实时读写的数据库系统。
- 为Spark和Hadoop MapReduce提供海量数据实时处理能力。
- 利用ZooKeeper作为协同服务。

结构

HBase集群由主备Master进程和多个RegionServer进程组成。如图2-9所示。

图 2-9 HBase 结构

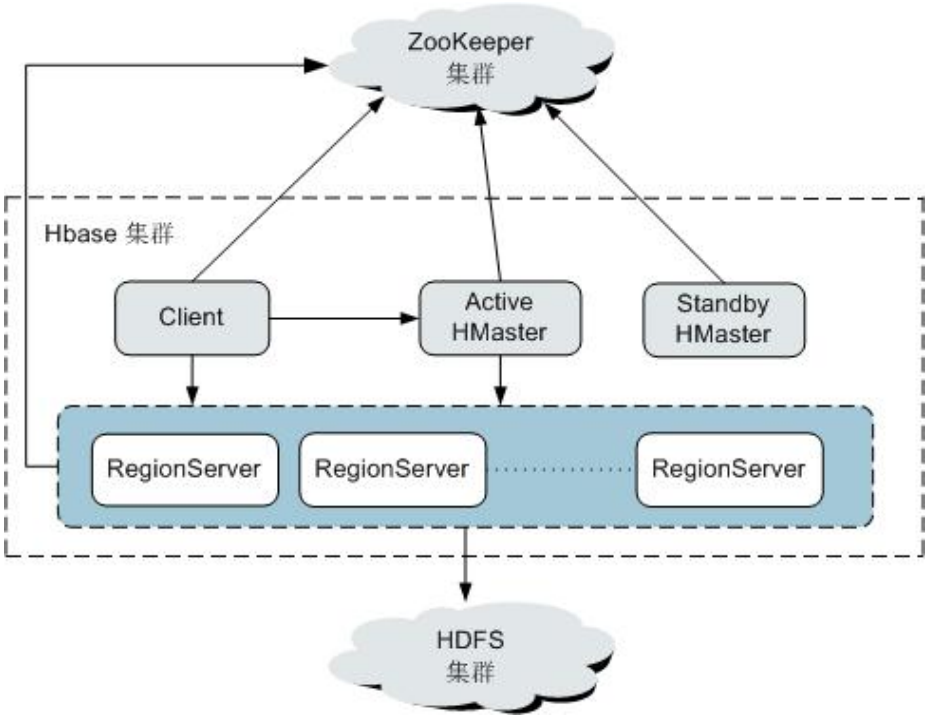


表 2-4 模块说明

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将自己的信息注册到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。

原理

- **HBase数据模型**

  HBase以表的形式存储数据，数据模型如图2-10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Row Key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。您可以在创建表时定义Region的Row Key区间，或者在配置文件中定义Region的大小。

  图 2-10 HBase 数据模型

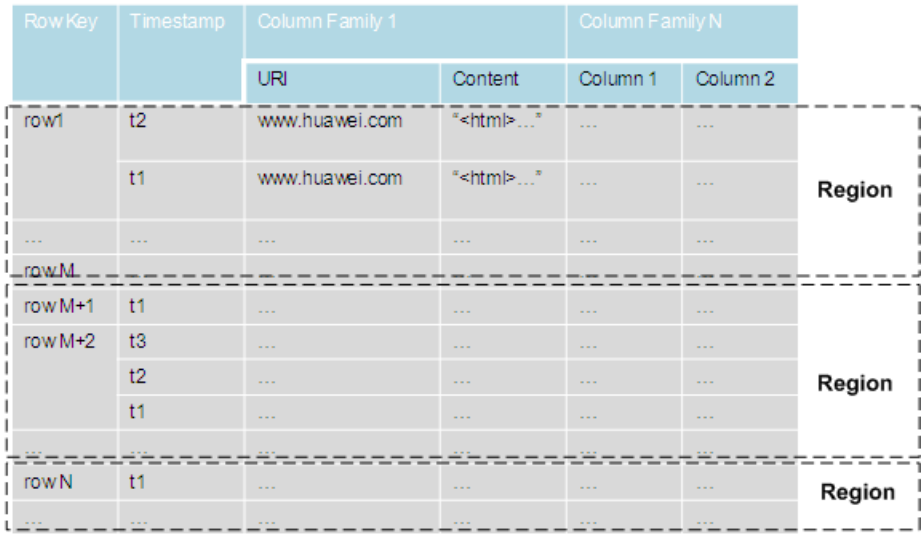


表 2-5 概念介绍

名称	描述
Row Key	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Row Key。所有记录按照Row Key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。

名称	描述
Column	列，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2-11所示。

  图 2-11 RegionServer 的数据存储结构

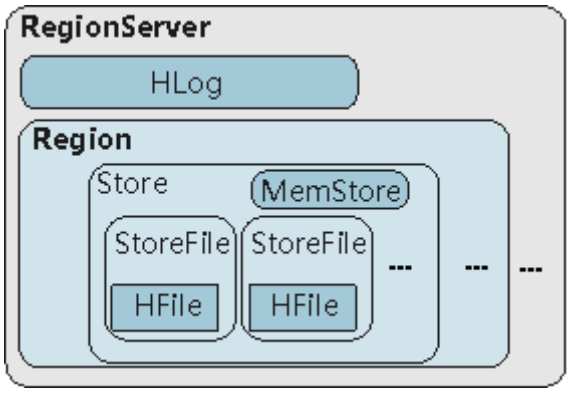
  

  图2-11中Region的各部分的说明如表2-6所示。

  表 2-6 Region 结构说明

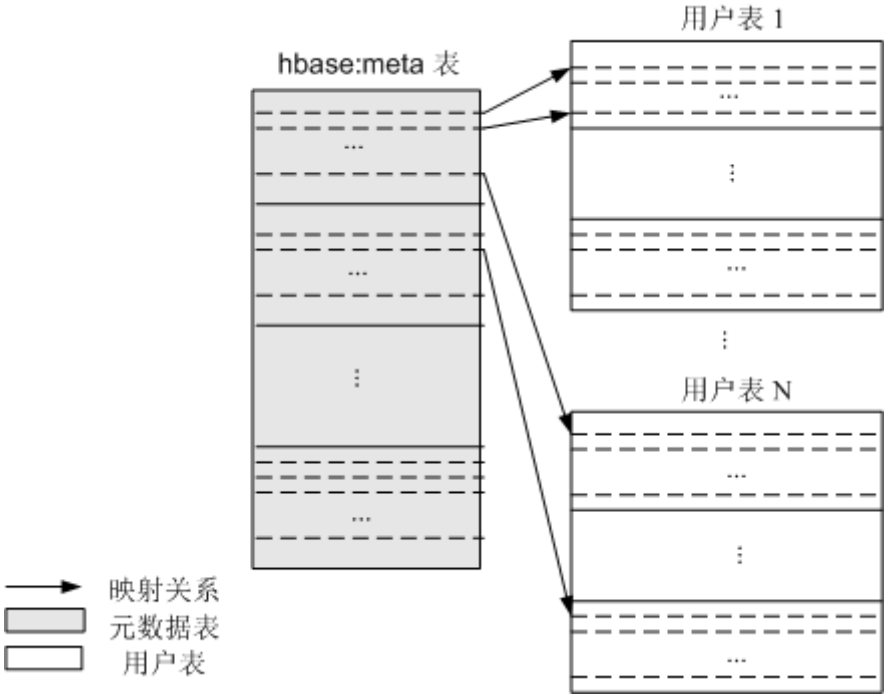
名称	描述
Store	一个Region由一个或多个Store组成，每个Store对应图2-10中的一个Column Family。
MemStore	一个Store包含一个MemStore，MemStore缓存客户端向Region插入的数据，当RegionServer中的MemStore大小达到配置的容量上限时，RegionServer会将MemStore中的数据“flush”到HDFS中。
StoreFile	MemStore的数据flush到HDFS后成为StoreFile，随着数据的插入，一个Store会产生多个StoreFile，当StoreFile的个数达到配置的最大值时，RegionServer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，用来记录用户表的Region信息，例如，Region位置、起始Row Key及结束Row Key等信息。

  元数据表和用户表的映射关系如图2-12所示。

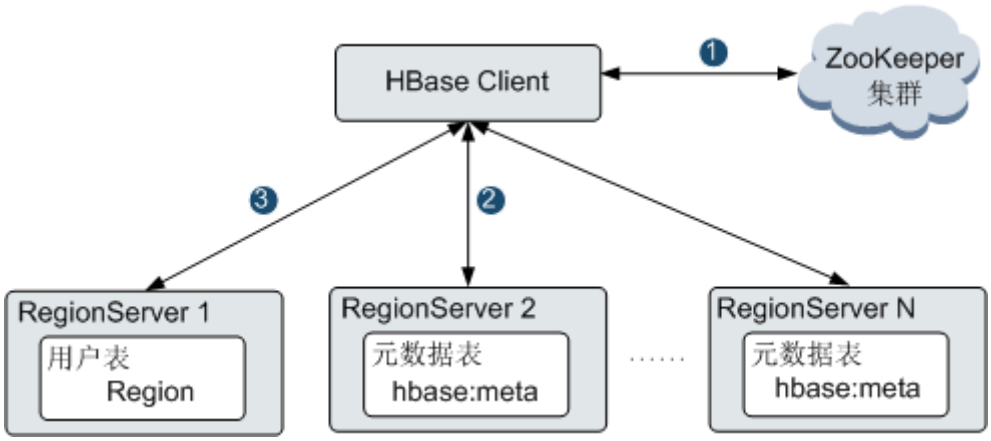
图 2-12 元数据表和用户表的映射关系



- **数据操作流程**

  HBase数据操作流程如图2-13所示。

图 2-13 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。

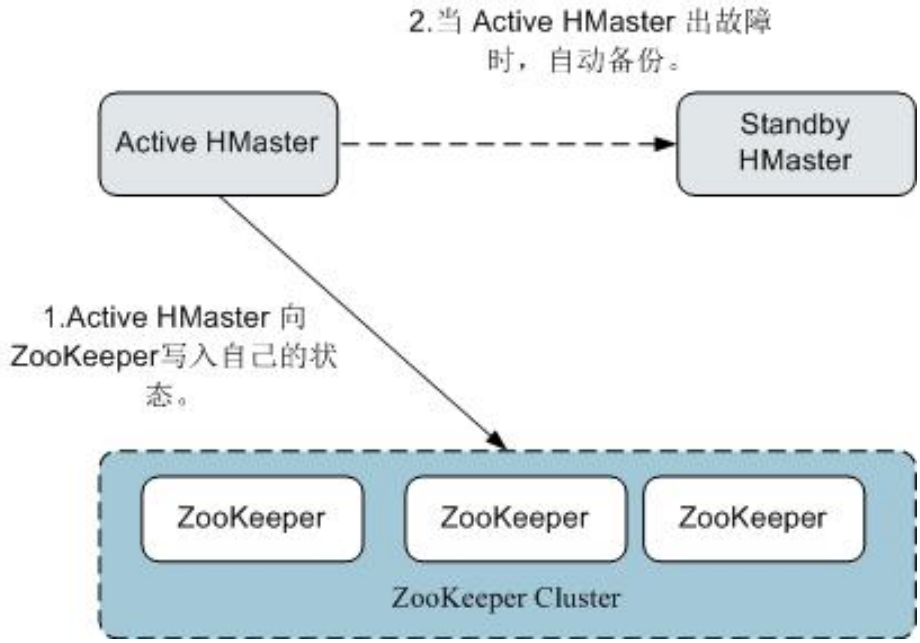
2.3.3.2 HA 方案介绍

背景

HBase中的HMaster负责region分配，维护meta信息表，当regionserver服务停止后把相应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。

实现方案

图 2-14 HMaster 高可用性实现架构



HMaster高可用性架构通过在ZooKeeper集群创建empheral zookeeper node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节点添加监听事件。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节点通过监听事件（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创建监听事件。

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。

2.3.3.3 与组件的关系

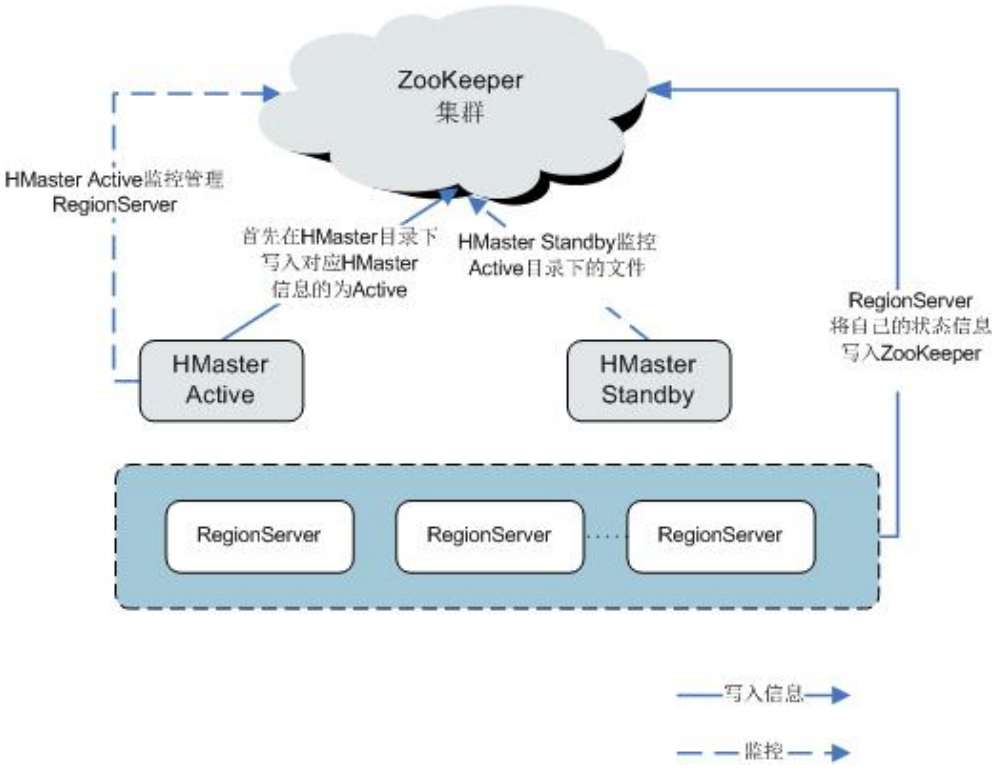
HDFS 和 HBase 的配合关系

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。

ZooKeeper 和 HBase 的配合关系

ZooKeeper与HBase的关系如图2-15所示。

图 2-15 ZooKeeper 和 HBase 的关系



1. HRegionServer把自己以Ephemeral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。

2.3.4 HDFS

2.3.4.1 基本原理

简介

Hadoop分布式文件系统（Hadoop Distributed File System）能提供高吞吐量的数据访问，适合大规模数据集方面的应用，为海量数据提供存储。

结构

HDFS包含主、备NameNode和多个DataNode，如图2-16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 2-16 HA HDFS 结构

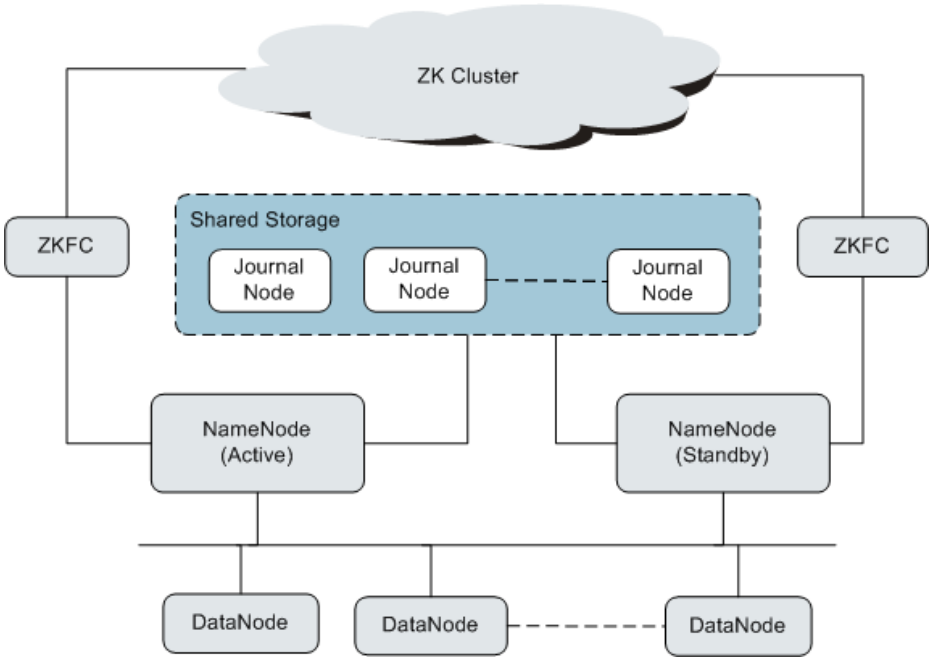


图2-16中各模块的功能说明如表2-7所示。

表 2-7 模块说明

名称	描述
Name Node	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。

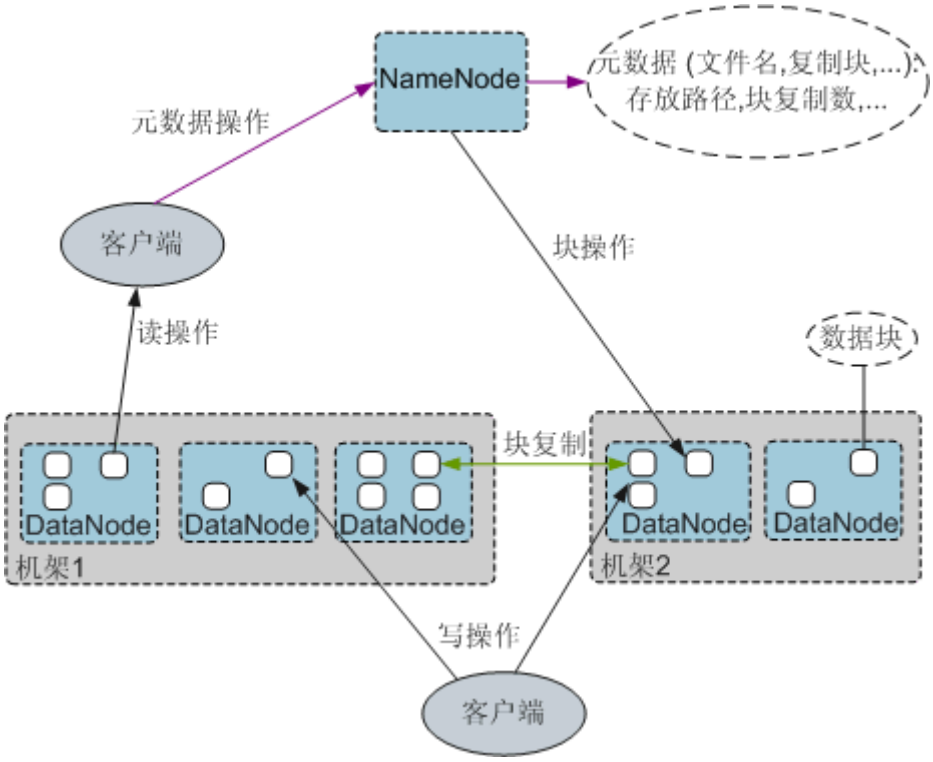
名称	描述
DataNode	用于存储每个文件的“数据块”数据，并且会周期性地向NameNode报告该DataNode的数据存放情况。
JournalNode	HA集群下，用于同步主备NameNode之间的元数据信息。
ZKFC	ZKFC是需要和NameNode一一对应的服务，即每个NameNode都需要部署ZKFC。它负责监控NameNode的状态，并及时把状态写入Zookeeper。ZKFC也有选择谁作为Active NameNode的权利。
ZK Cluster	ZooKeeper是一个协调服务，帮助ZKFC执行主NameNode的选举。

原理

- **HDFS原理**

  HDFS的原理如图2-17所示。

  图 2-17 HDFS 原理



在HDFS内部，一个文件分成一个或多个“数据块”，这些“数据块”存储在DataNode集合里，NameNode负责保存和管理所有的HDFS元数据。客户端连接到NameNode，执行文件系统的“命名空间”操作，例如打开、关闭、重命名文件和目录，同时决定“数据块”到具体DataNode节点的映射。DataNode在NameNode的指挥下进行“数据块”的创建、删除和复制。客户端连接到DataNode，执行读写数据块操作。

- **HDFS HA架构**

  HA即为High Availability，用于解决NameNode单点故障问题，该特性通过热备的方式为主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。

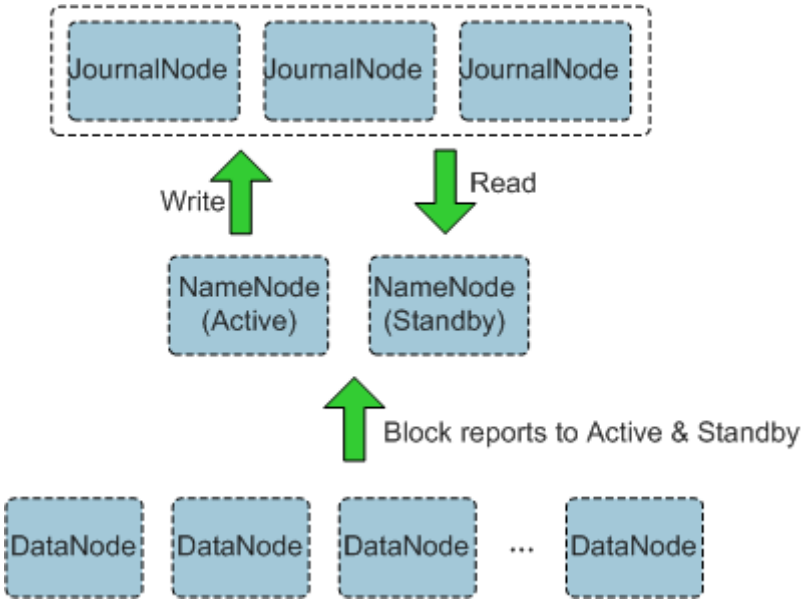
  在一个典型HDFS HA场景中，通常由两个NameNode组成，一个处于Active状态，另一个处于Standby状态。

  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2-18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部署在管理节点上，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 2-18 基于 QJM 的 HDFS 架构



2.3.4.2 HA 方案介绍

背景

在Hadoop2.0.0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。

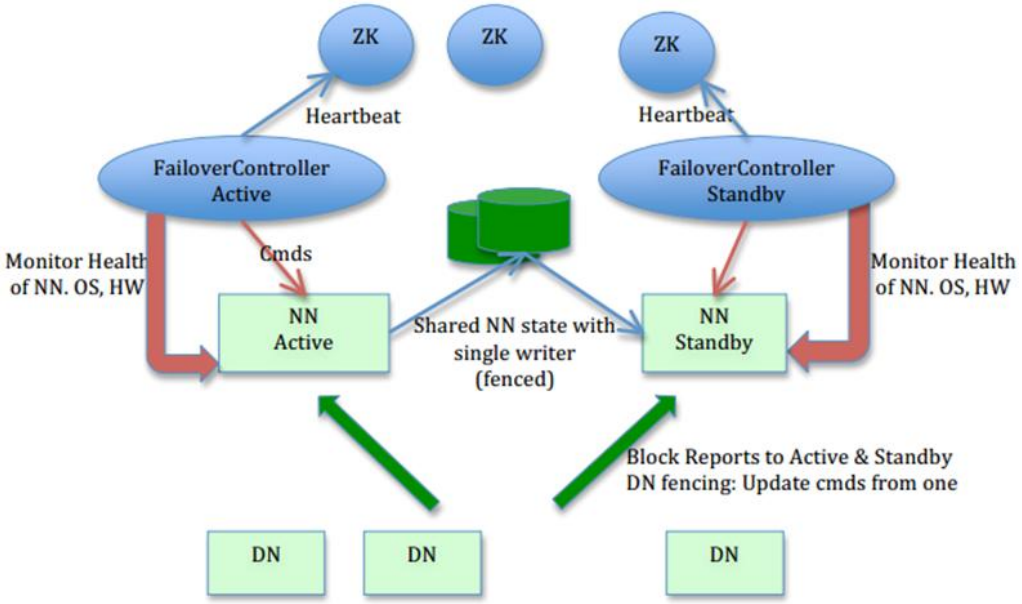
（2）计划性的维护工作，如软硬件升级等，将导致集群停止工作。

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。当一台机器崩溃时，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。

实现方案

图 2-19 典型的 HA 部署方式



在一个典型的HA集群中（如图2-19），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节点能够确保有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：

http://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html

2.3.4.3 与组件的关系

HDFS 和 HBase 的配合关系

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。

MapReduce 和 HDFS 的配合关系

- 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。

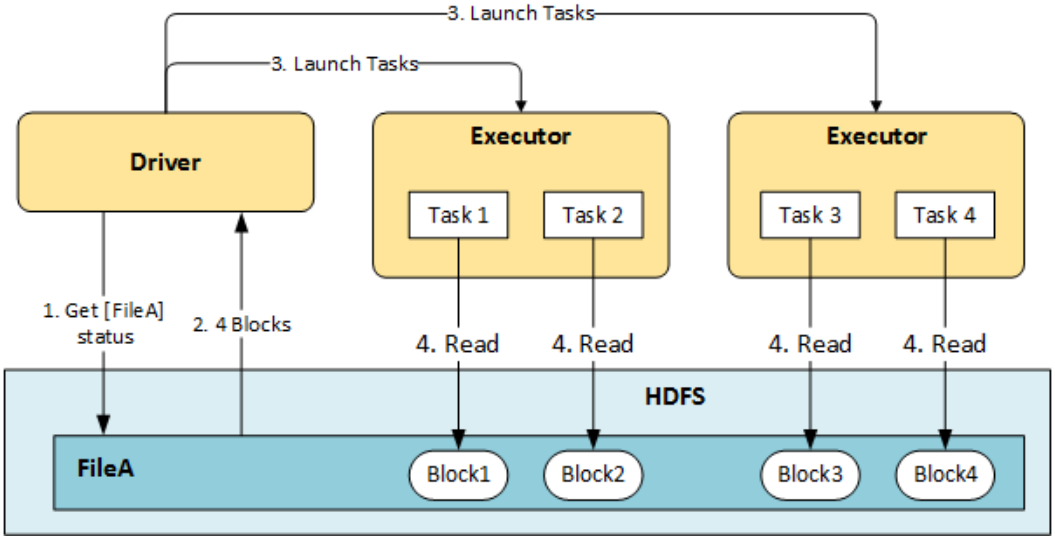
Spark 和 HDFS 的配合关系

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2-20所示。

图 2-20 读取文件过程



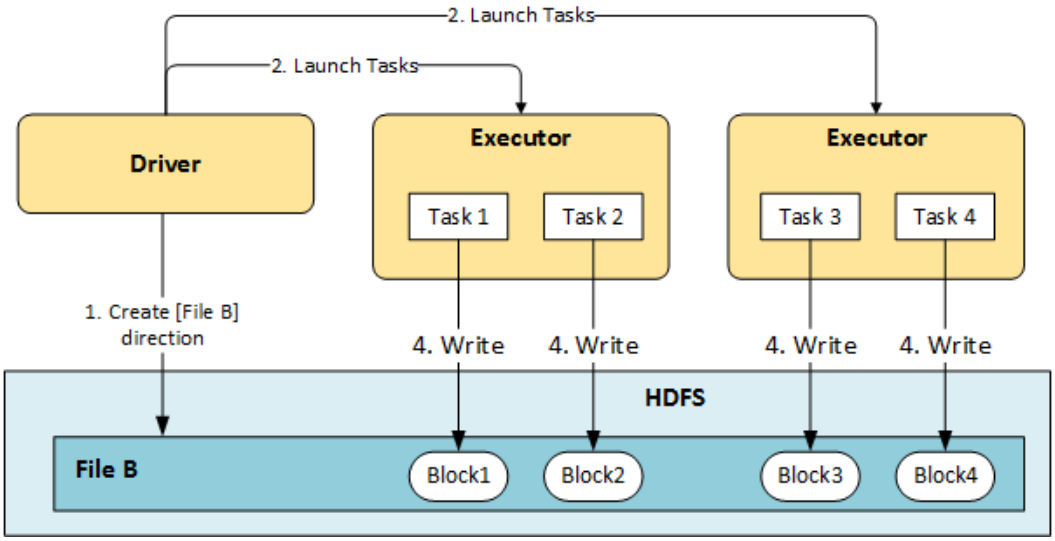
读取文件步骤的详细描述如下所示：

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2-21所示。

图 2-21 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。

SmallFS 和 HDFS 的配合关系

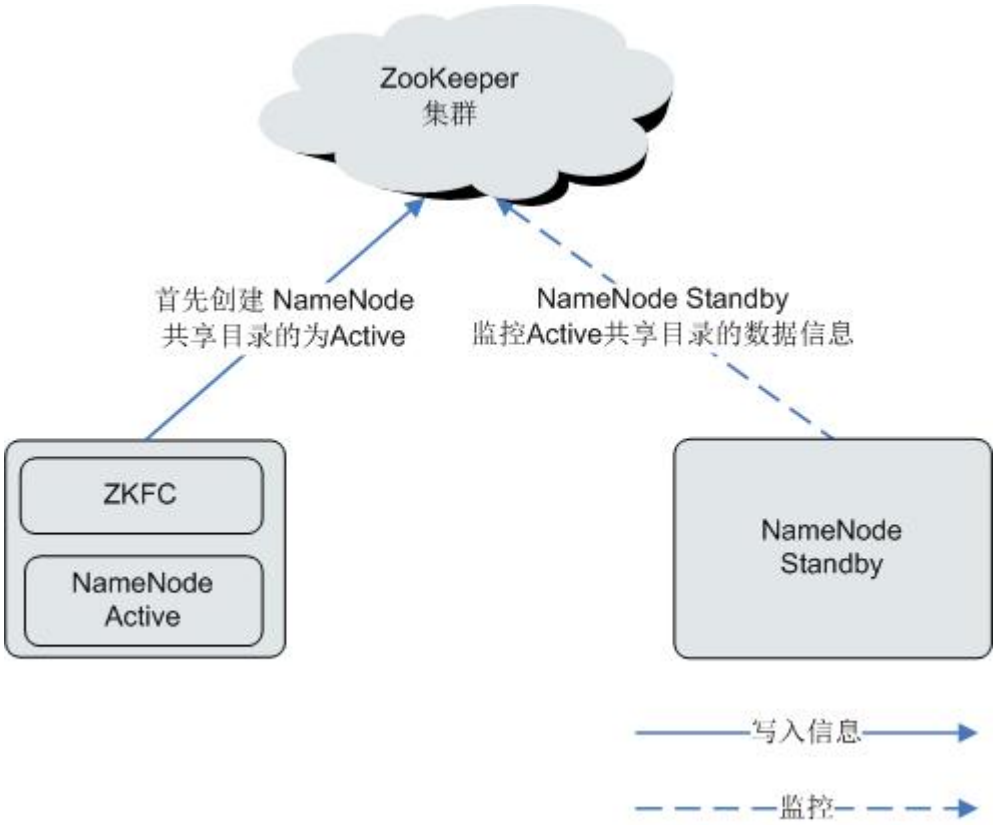
SmallFS是构建于HDFS上层的文件系统，可以对HDFS上的小文件进行合并，从而解决HDFS上面由于过多的小文件对系统造成冲击的问题，并且为客户提供透明化的小文件操作接口。

HDFS上面的大量小文件作为SmallFS的输入，合并后的大文件输出到HDFS文件系统。

ZooKeeper 和 HDFS 的配合关系

ZooKeeper与HDFS的关系如图2-22所示。

图 2-22 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。

2.3.5 SmallFS

2.3.5.1 基本原理

简介

HDFS中NameNode负责管理整个文件系统的元数据。由于NameNode的元数据将使用内存来管理，当产品业务在使用中生成较多的小文件时，会迅速消耗掉NameNode上的内存，对NameNode运行产生压力，影响整个集群的效率。

针对该业务场景，华为大数据新增小文件后台合并特性（即SmallFS服务），将大量小文件按一定的周期进行合并、删除、清理任务来减少HDFS上的小文件数量，大大降低NameNode元数据管理压力。

结构

FGCService是小文件特性的主进程，小文件特性又称为SmallFS。

如图2-23所示，在HA模式下，集群中有两个FGCService进程，其中Active FGCService作为小文件系统的主进程，在后台调度启用Merge、Delete、Cleanup任务。Standby FGCService对元数据做checkpoint，存储在LevelDB中。SmallFS Client通过RPC协议调用接口访问FGCService来操作小文件。

后台调度的任务详情，如下所示：

- Merge任务：FGCService定期通过MapReduce任务将HDFS上的小文件合并成大文件存储到HDFS上，将元数据存储到LevelDB并删除合并的小文件。
- Delete任务：FGCService定期通过MapReduce任务读取LevelDB的元数据重新清理合并的大文件，将已删除的小文件的内容从大文件中删除。
- Cleanup任务：在FGCService服务出现停止、意外退出等不正常的情况下，清理Merge或者Delete任务失败的文件。

图 2-23 SmallFS 架构

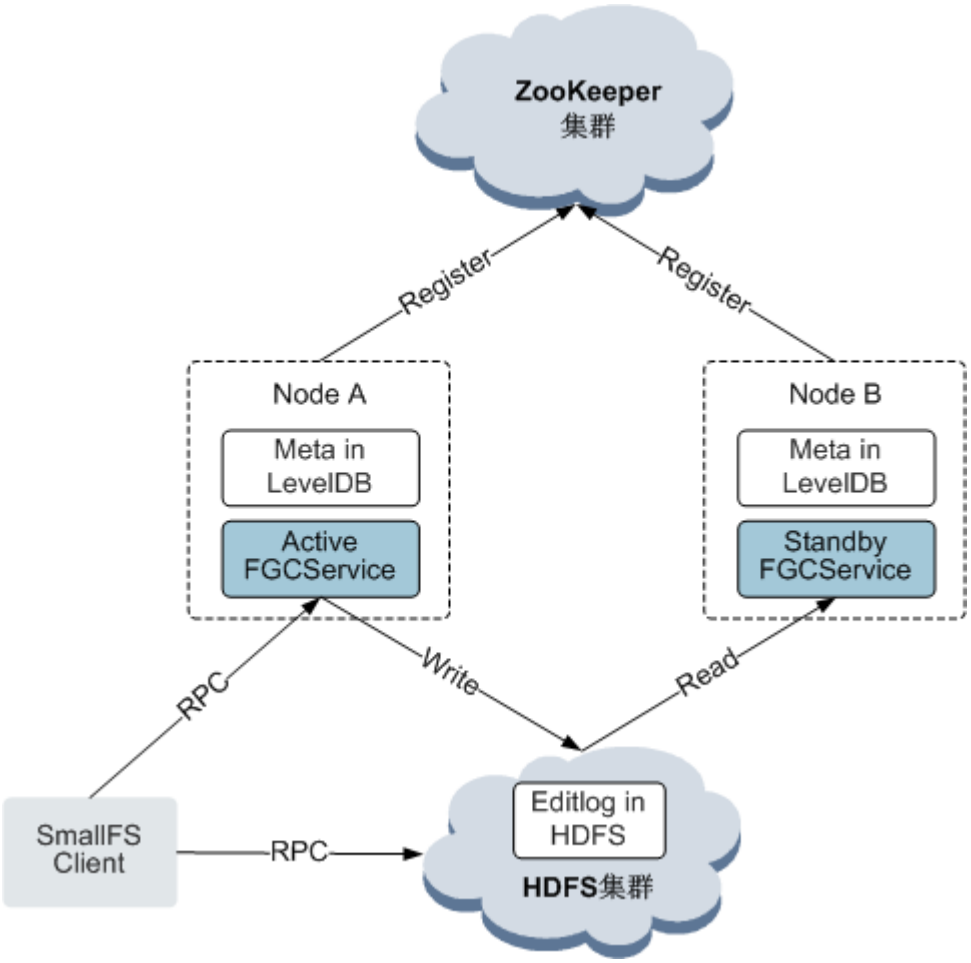


表 2-8 模块介绍

模块	描述
SmallFS Client	SmallFS服务客户端。
FGCService	SmallFS的主进程，HA模式下包含Active和Standby，一旦某一个节点故障，可迅速进行主备切换，保证SmallFS服务正常。
LevelDB	用于管理SmallFS元数据，集群通过HDFS Editlog保证主备节点中元数据的一致性。
Editlog	记录Active FGCService对元数据的操作。

2.3.5.2 HA 方案介绍

背景

FGC HA模式下，在两个独立的机器中运行了两个FGCService，一个为主节点，另一个为备节点。当一个节点由于机器、网络或进程失败等原因导致故障时，另一个节点还可以正常运行。

实现方案

在FGC HA模式下，2个FGCService运行在两个不同的节点中。一个为主节点，另一个为备节点。主节点用于处理客户端操作，例如合并、删除、清除等操作。备节点则作为从节点，必要时提供快速故障转移和恢复的能力。

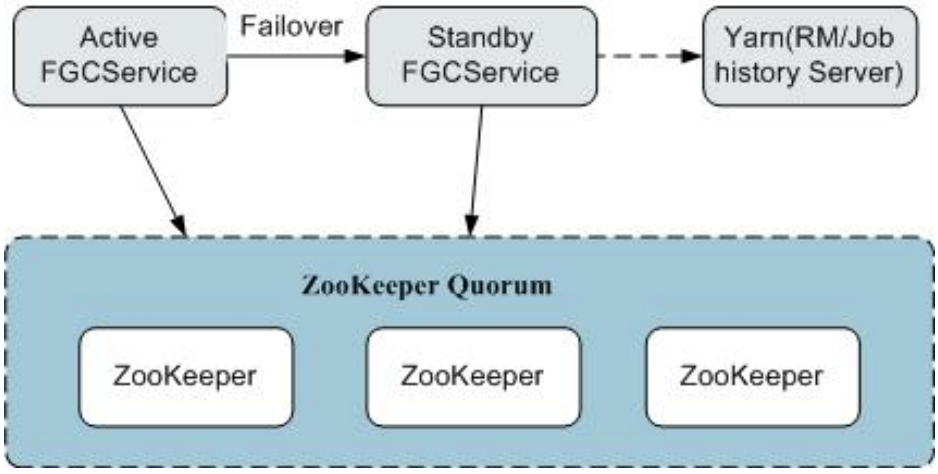
当namespace的修改在主节点中执行时，此修改的日志会被保存在HDFS目录的Editlog文件中。备节点会不断关注此目录的编辑，当发现出现修改时，会将修改应用到备节点自己的namespace中。当发生故障时，备节点可以保证在切换为主节点之前，HDFS存储的所有修改都已记录。保证了namespace状态的全量同步。

为支持故障恢复，FGCService使用如下hadoop组件：

1.Zookeeper

2.Job History Server

图 2-24 FGC HA Service



2.3.5.3 与组件的关系

SmallFS 和 HDFS 的配合关系

SmallFS是构建于HDFS上层的文件系统，可以对HDFS上的小文件进行合并，从而解决HDFS上面由于过多的小文件对系统造成冲击的问题，并且为客户提供透明化的小文件操作接口。

HDFS上面的大量小文件作为SmallFS的输入，合并后的大文件输出到HDFS文件系统。

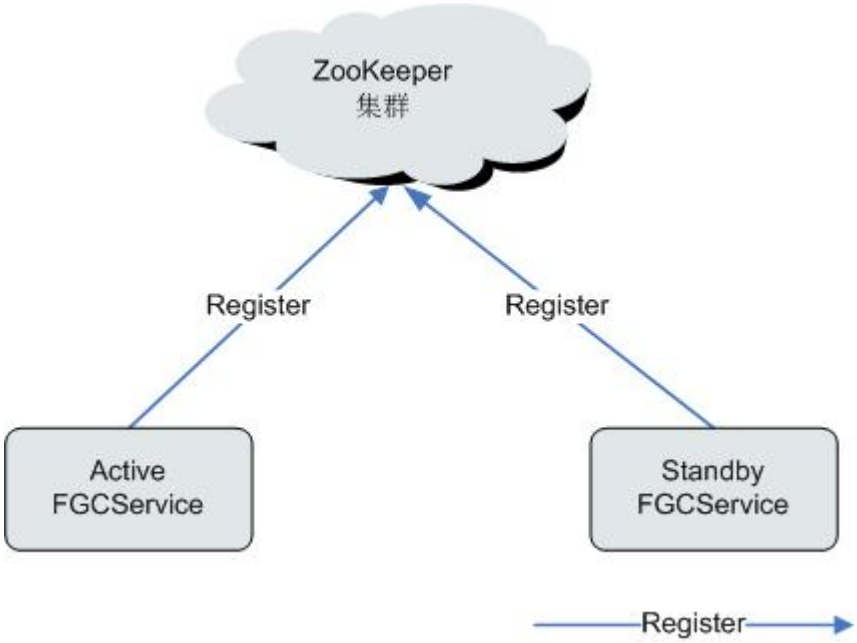
SmallFS 和 YARN 的配合关系

SmallFS将会定期运行merge、delete、cleanup任务，而这些任务是在YARN上运行MapReduce任务来对HDFS上的文件数据进行merge、delete和cleanup操作。

SmallFS 和 Zookeeper 的配合关系

FGCService的部署模式为HA模式。HA（High Availability）模式目的是防止单节点故障导致服务不可用。为了支持HA模式，FGCService依赖于ZooKeeper。

图 2-25 ZooKeeper 和 SmallFS 的关系



2.3.6 Yarn

2.3.6.1 基本原理

简介

YARN是Hadoop2.0中的资源管理系统，它是一个通用的资源管理模块，可以为各类应用程序进行资源管理和调度。YARN不仅局限于MapReduce一种框架使用，也可以供其他框架使用，比如Tez、Spark、Storm等。YARN类似于资源管理系统Mesos和更早的Torque。

结构

YARN模型主要由ResourceManager、ApplicationMaster和NodeManager组成，如下图2-26所示。

图 2-26 Apache YARN 的基本架构

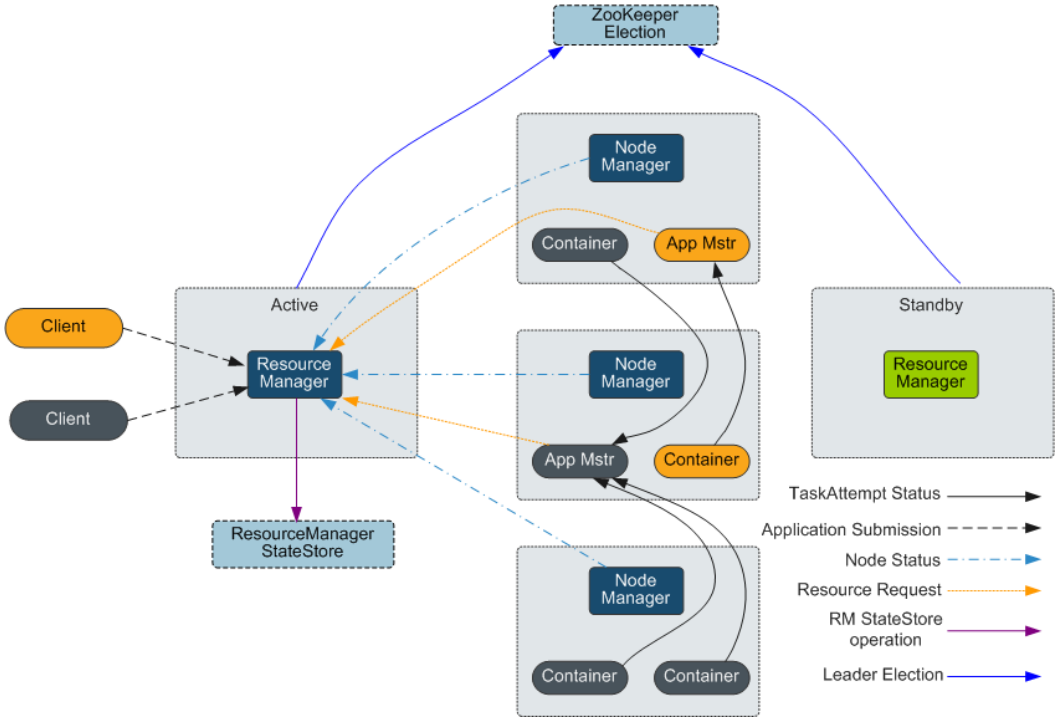


图2-26中各部分的功能如表2-9所示。

表 2-9 结构图说明

名称	描述
Client	YARN Application客户端，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。它接收来自各个节点（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，它管理Hadoop集群中单个计算节点，包括与ResourceManger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。

名称	描述
Container	Container是YARN中的资源抽象，它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。

原理

新的Hadoop MapReduce框架命名为MRv2或者叫YARN。YARN主要分为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。它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，它将内存、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它等。
- NodeManager：NM是每个节点上的资源和任务管理器，一方面，它会定时向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，它接收并处理来自AM的Container启动/停止等各种请求。
- 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。

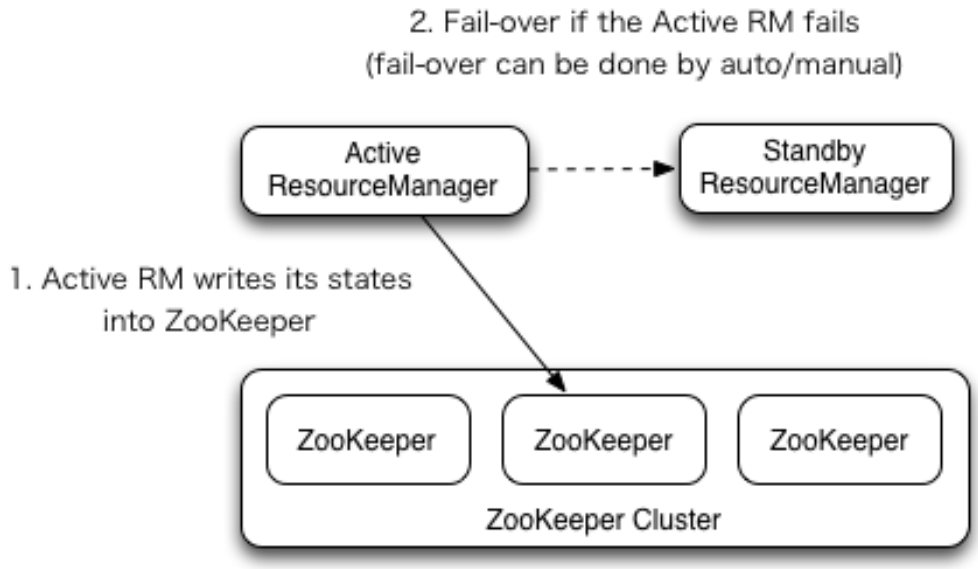
2.3.6.2 HA 方案介绍

背景

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。

实现方案

图 2-27 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2-27）。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，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ResouceManager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**ResourceManger 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：

http://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/ResourceManagerHA.html

2.3.6.3 与组件的关系

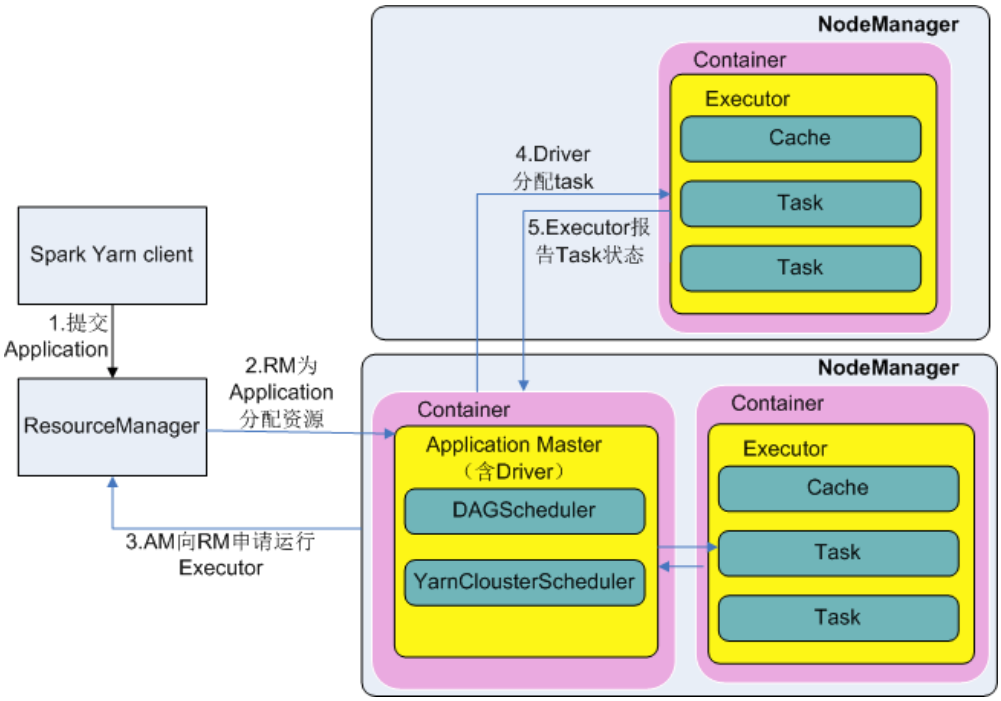
Spark 和 YARN 的配合关系

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2-28所示。

  图 2-28 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。

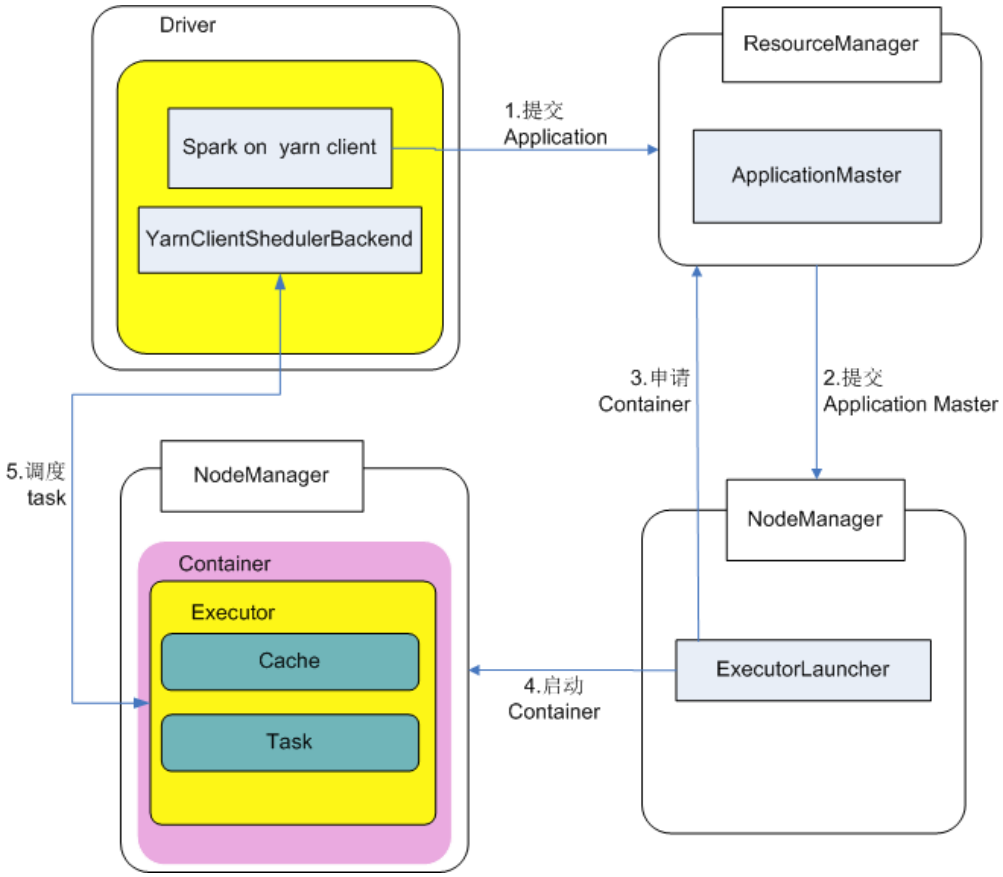
  d. Driver分配Task给Executor执行。

  e. Executor执行Task并向Driver汇报运行状况。

- YARN Client模式

  运行框架如图2-29所示。

图 2-29 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。

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。

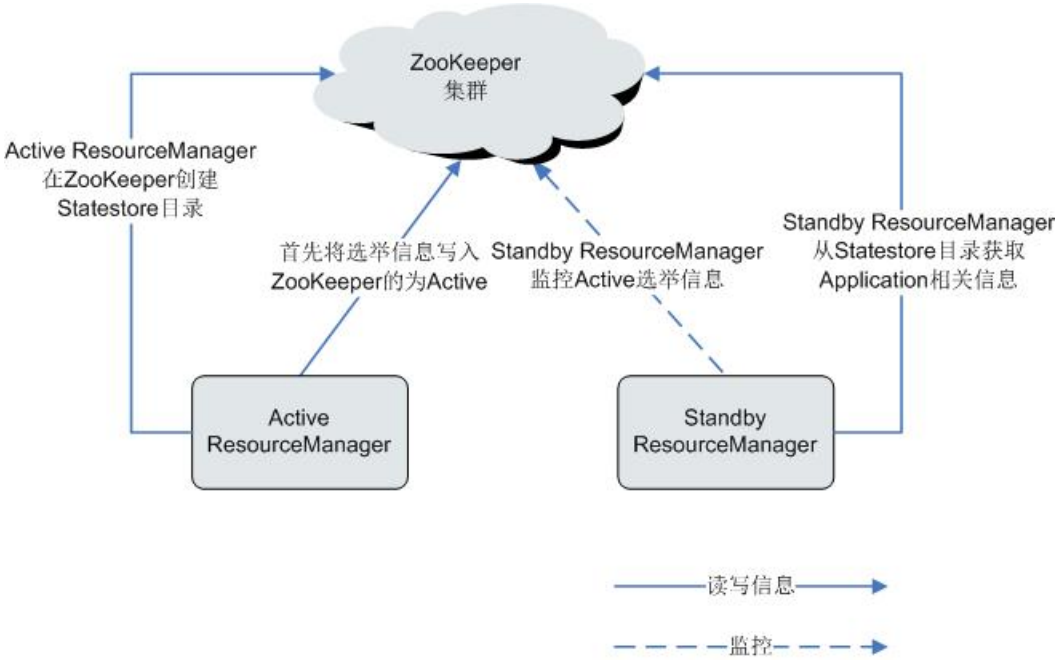
MapReduce 和 YARN 的配合关系

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。

ZooKeeper 和 YARN 的配合关系

ZooKeeper与YARN的关系如图2-30所示。

图 2-30 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，第一个成功把写入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。

SmallFS 和 YARN 的配合关系

SmallFS将会定期运行merge、delete、cleanup任务，而这些任务是在YARN上运行MapReduce任务来对HDFS上的文件数据进行merge、delete和cleanup操作。

2.3.7 Mapreduce

2.3.7.1 基本原理

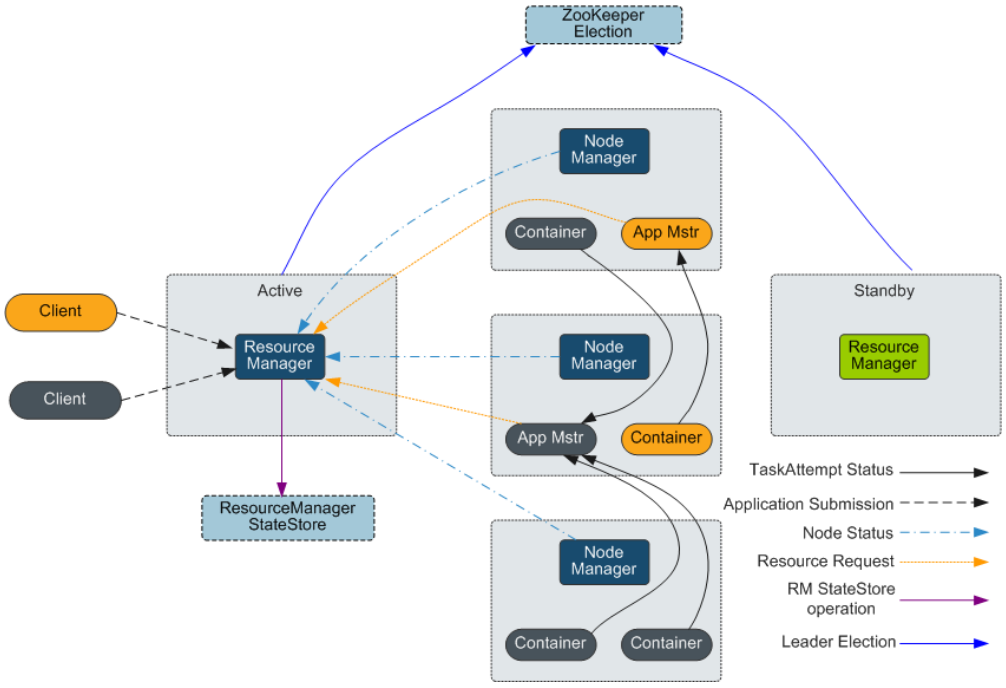
简介

MapReduce是一种简化并行计算的编程模型，名字源于该模型中的两项核心操作：Map和Reduce。Map将一个作业分解成为多个任务，Reduce将分解后多个任务处理的结果汇总起来，得出最终的分析结果。

结构

如图2-31所示，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。

图 2-31 Apache YARN&MapReduce 的基本架构



2.3.7.2 与组件的关系

MapReduce 和 HDFS 的配合关系

- 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。

MapReduce 和 YARN 的配合关系

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。

2.3.8 Spark

2.3.8.1 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。

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、MLlib、GraphX等形成数据处理栈，提供一站式数据处理能力。
- 完美契合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2-32所示，各模块的说明如表2-10所示。

图 2-32 Spark 架构

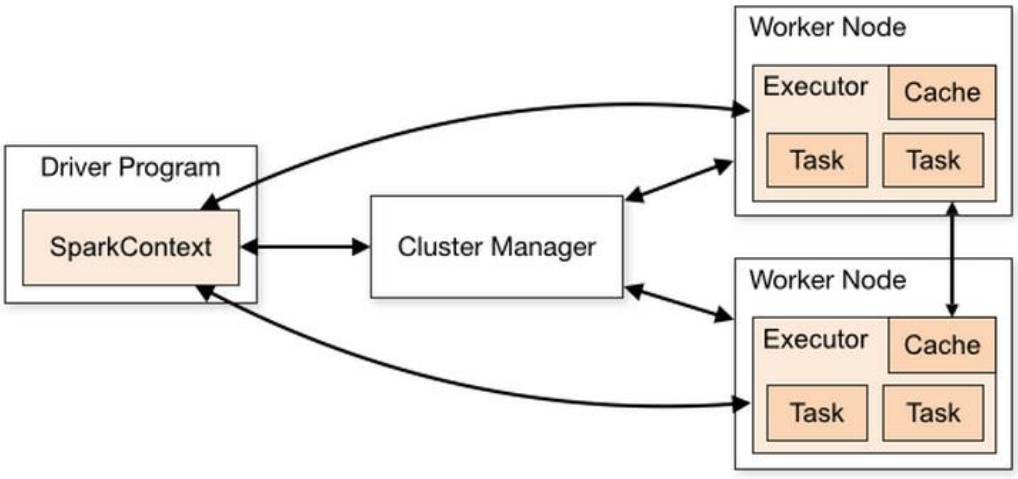


表 2-10 基本概念说明

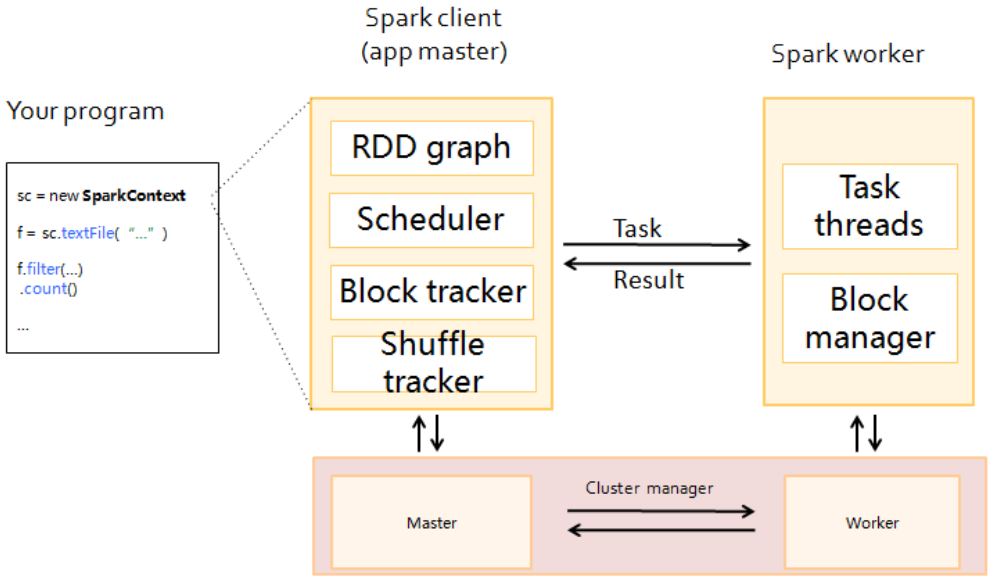
模块	说明
Cluster Manager	集群管理器，管理集群中的资源。Spark支持多种集群管理器，Spark自带的Standalone集群管理器、Mesos或YARN。华为Spark集群默认采用YARN模式。
Application	Spark应用，由一个Driver Program和多个Executor组成。
Deploy Mode	部署模式，分为cluster和client模式。cluster模式下，Driver会在集群内的节点运行；而在client模式下，Driver在客户端运行（集群外）。
Driver Program	是Spark应用程序的主进程，运行Application的main()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平台中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。

原理

Spark的应用运行架构如**图2-33**所示，运行流程如下所示：

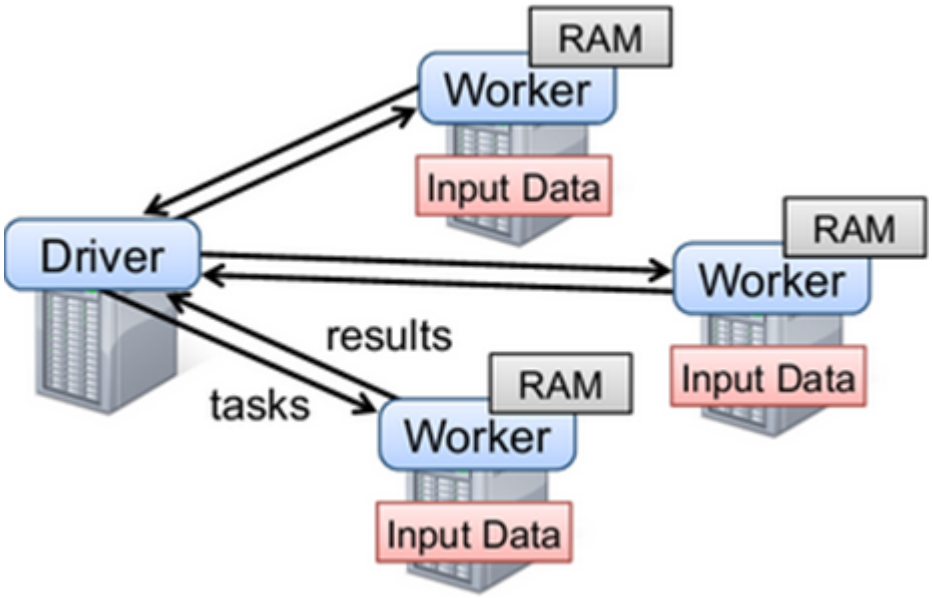
1. 应用程序（Application）是作为一个进程的集合运行在集群上的，由Driver进行协调。
2. 在运行一个应用时，Driver会去连接集群管理器（Standalone、Mesos、YARN）申请运行Executor资源，并启动ExecutorBackend。然后由集群管理器在不同的应用之间调度资源。Driver同时会启动应用程序DAG调度、Stage划分、Task生成。
3. 然后Spark会把应用的代码(传递给SparkContext的JAR或者Python定义的代码)发送到Executor上。
4. 所有的Task执行完成后，用户的应用程序运行结束。

图 2-33 Spark 应用运行架构



Spark采用Master和worker的模式，如图2-34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 2-34 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程序最好和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。

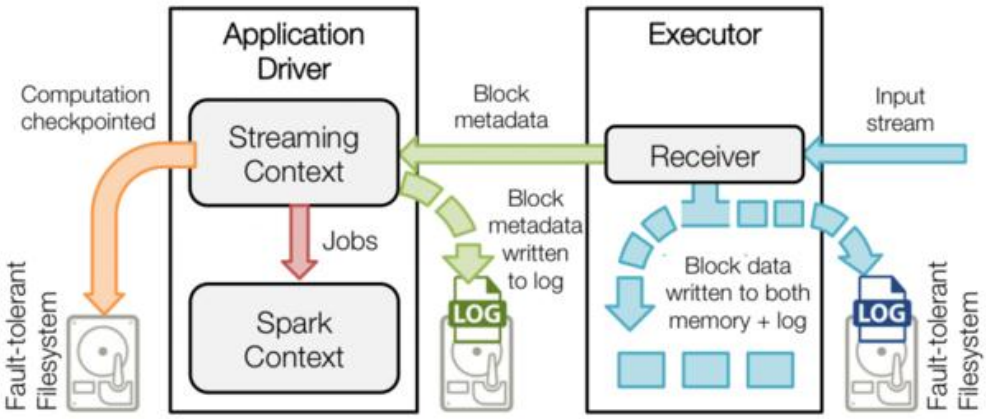
Spark Streaming 原理

Spark Streaming是一种构建在Spark上的实时计算框架，它扩展了Spark处理大规模流式数据的能力。

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2-35所示：

图 2-35 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。
   - 块数据在日志中的偏移信息（如果启用了WAL，即预写日志功能）。

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和它们的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和它的RDD抽象设计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的长正常运行时间需求，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，因为它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或S3这样的容错文件系统中了。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，它们就有可能会丢失。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，连同它们在内存中的所有数据，也同时被终止。对于Spark Streaming来说，从诸如Kafka和Flume的数据源接收到的所有数据，在它们处理完成之前，一直都缓存在Executor的内存中。纵然Driver重新启动，这些缓存的数据也不能被恢复。为了避免这种数据损失，Spark Streaming引进了预写日志（Write Ahead Logs）功能。

预写日志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。这个操作的思想是首先将操作记入一个持久的日志，然后才对数据施加这个操作。假如在施加操作的中间系统失败了，通过读取日志并重新施加前面预定的操作，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。

像Kafka这样的数据源使用Receiver来接收数据。它们作为长驻运行任务在Executor中运行，负责从数据源接收数据，并且在数据源支持时，还负责确认收到的数据。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。因此即使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，它既用作保存流的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。

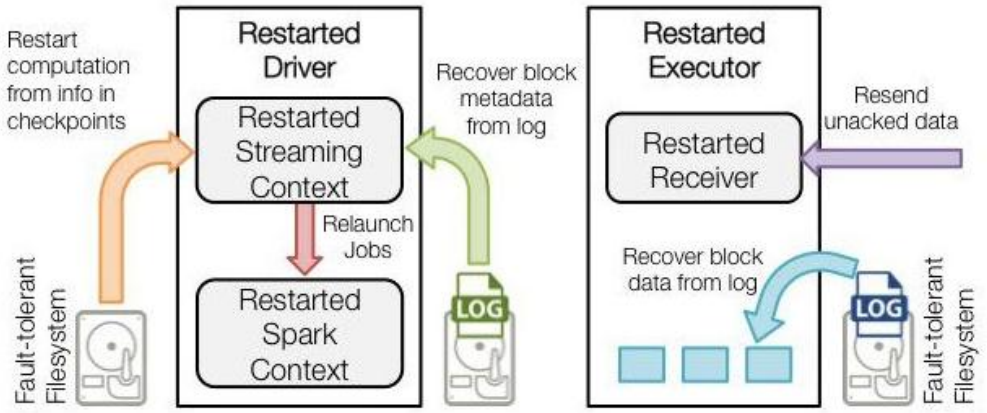
说明

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，最好创建更多的Recevier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 2-36 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）

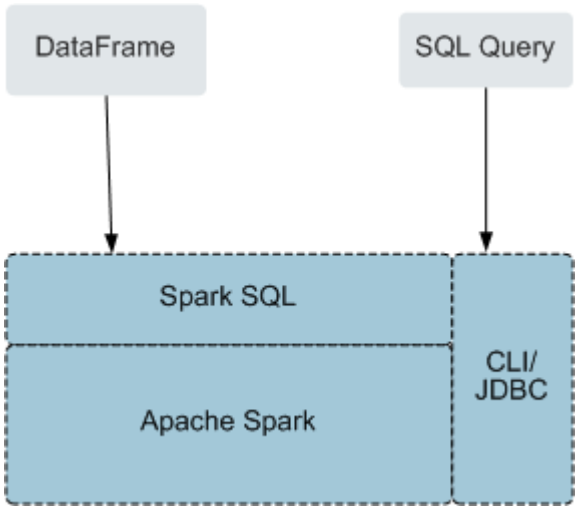
   失败时没有保存到日志中的缓存数据将由数据源再次发送。因为Receiver尚未对其确认。

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 DataFrame 原理

SparkSQL

图 2-37 SparkSQL 和 DataFrame



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

Spark SQL以及DataFrame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、Avro、Parquet、ORC、JSON和JDBC数据源，这些不同的数据源直接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。

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。

Spark SQL的程序入口是SQLContext类（或其子类），创建SQLContext时需要一个SparkContext对象作为其构造参数。SQLContext其中一个子类是HiveContext，相较于其父类，HiveContext添加了HiveQL的parser、UDF以及读取存量Hive数据的功能等。但注意，HiveContext并不依赖运行时的Hive，只是依赖Hive的类库。

由SQLContext及其子类可以方便的创建SparkSQL中的基本数据集DataFrame，DataFrame向上提供多种多样的编程接口，向下兼容多种不同的数据源，比如Parquet、JSON、Hive数据、Database、HBase等，这些数据源都可以使统一的语法来读取。

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可供方便调试，JDBCServer提供JDBC接口来供外部直接发送JDBC请求来完成结构化数据的计算和解析。

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。

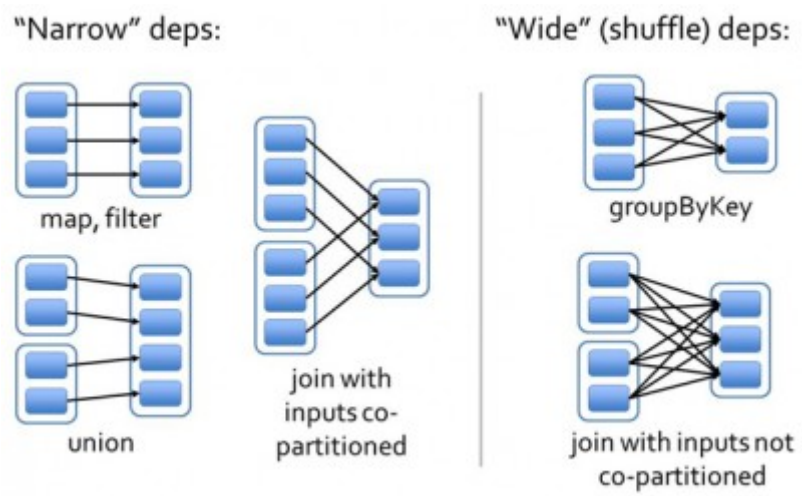
RDD的存储：

- 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 2-38 RDD 的依赖

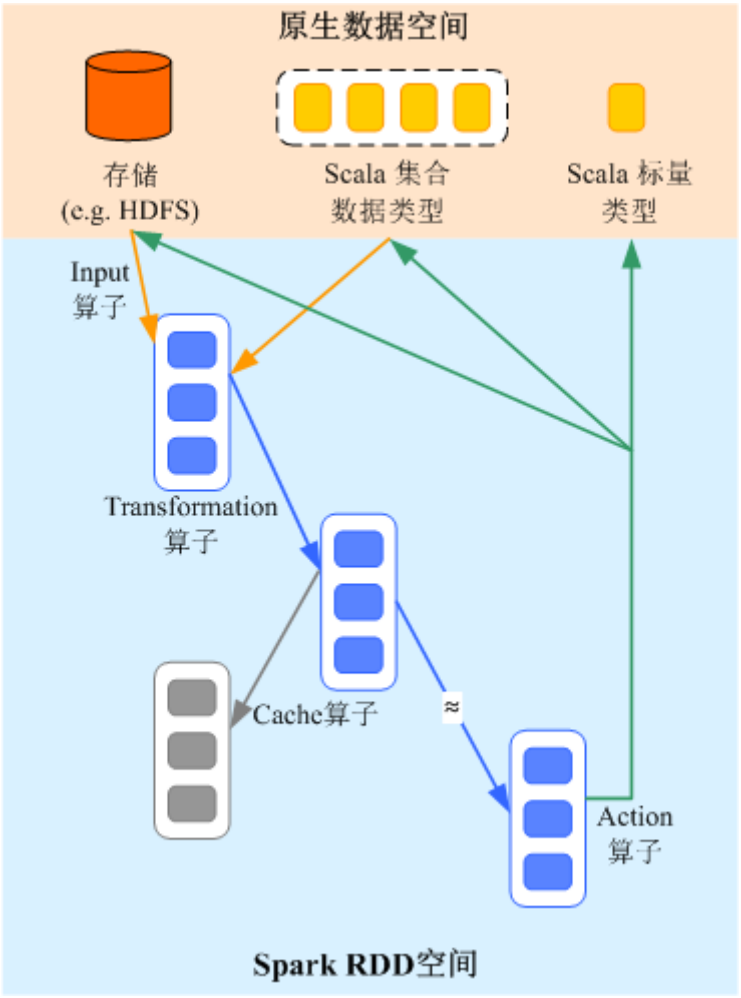
  - **窄依赖：**指父RDD的每一个分区最多被一个子RDD的分区所用。
  - **宽依赖：**指子RDD的分区依赖于父RDD的所有分区。

  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2-39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 2-39 RDD 操作示例



RDD看起来与Scala集合类型没有太大差别，但它们的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算子把它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、它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：

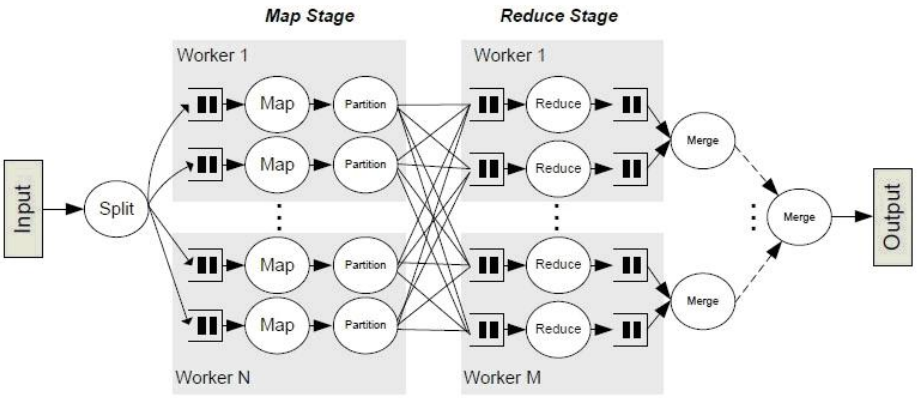
- 生成标量，如count（返回RDD中元素的个数）、reduce、fold/aggregate（返回几个标量）、take（返回前几个元素）。
- 生成Scala集合类型，如collect（把RDD中的所有元素倒入Scala集合类型）、lookup（查找对应key的所有值）。
- 写入存储，如与前文textFile对应的saveAsTextFile。
- 还有一个检查点算子checkpoint。当Lineage特别长时（这在图计算中时常发生），出错时重新执行整个序列要很长时间，可以主动调用checkpoint把当前数据写入稳定存储，作为检查点。

- **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，输出结果需要按key哈希，并且分发到每一个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 2-40 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为例讲一下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://spark.apache.org/docs/1.5.1/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。

2.3.8.2 HA 方案介绍

背景

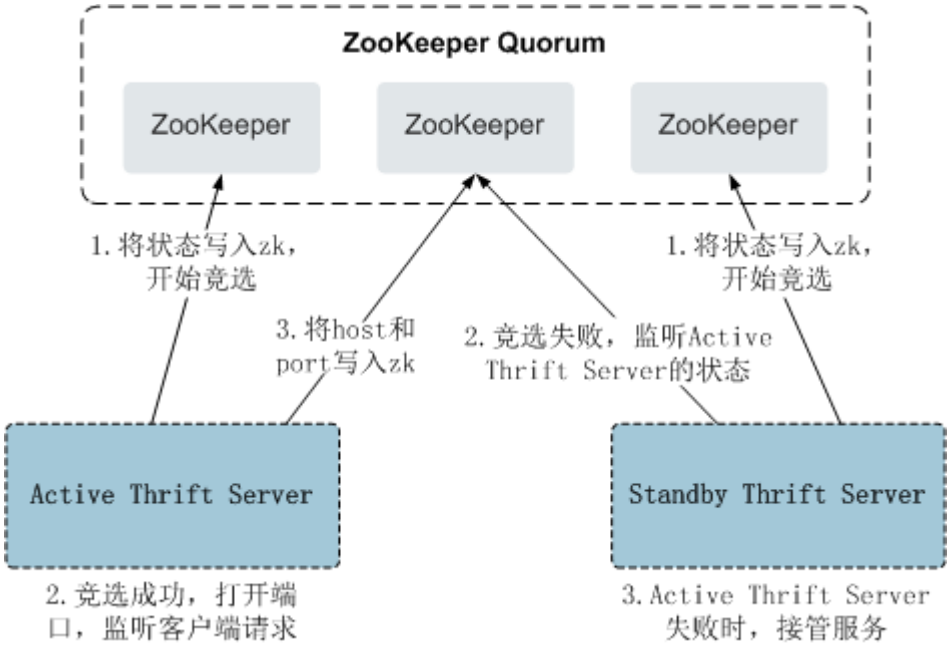
Spark中实现了类似于HiveServer2的Thrift JDBC服务，用户可以通过beeline以及JDBC接口访问。

基于社区已有的Thrift Server基础上，实现了其高可用性方案。

实现方案

基本原理如图所示：

图 2-41 Spark Thrift Server HA



Thrift Server进程在启动时，首先向Zookeeper注册自身消息，并开始竞选。若竞选为主，则打开端口监听来自客户端的请求，并将自身的host和port写入Zookeeper，供客户端连接使用；若竞选为备，则监听Zookeeper上主服务的状态，当主服务状态失败时接管服务。

Spark可以使用beeline来连接HA模式下的Thrift Server，连接字符串和非HA模式下的区别在于需要将ip:port替换为ha-cluster，例如：

```
jdbc:hive2://ha-cluster/default;user.principal=spark/hadoop.hadoop.com@HADOOP.COM;saslQop=auth-conf;auth=KERBEROS;principal=spark/hadoop.hadoop.com@HADOOP.COM;
```

其他使用方式和非HA模式下相同，请参见**https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients**

2.3.8.3 与组件的关系

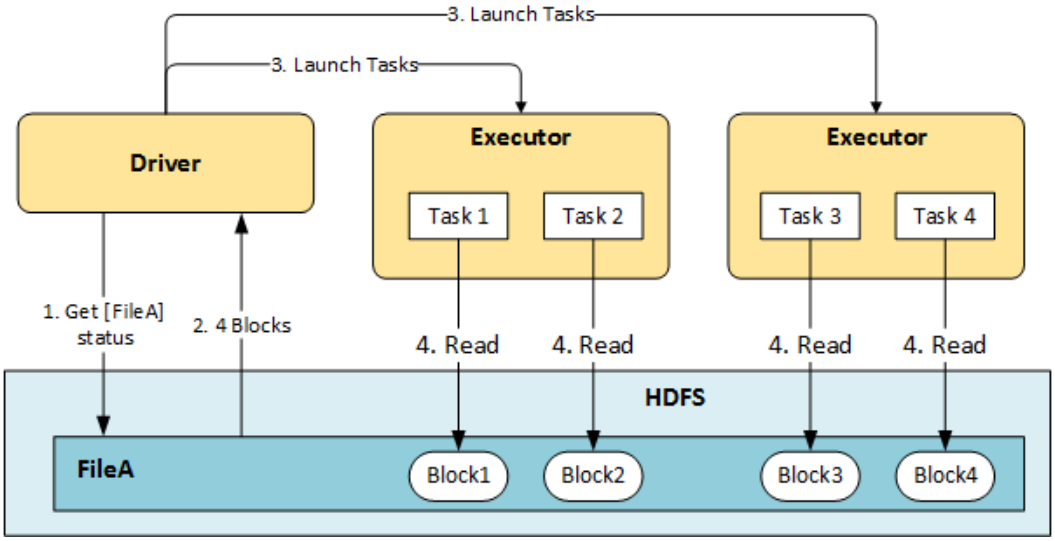
Spark 和 HDFS 的配合关系

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2-42**所示。

图 2-42 读取文件过程

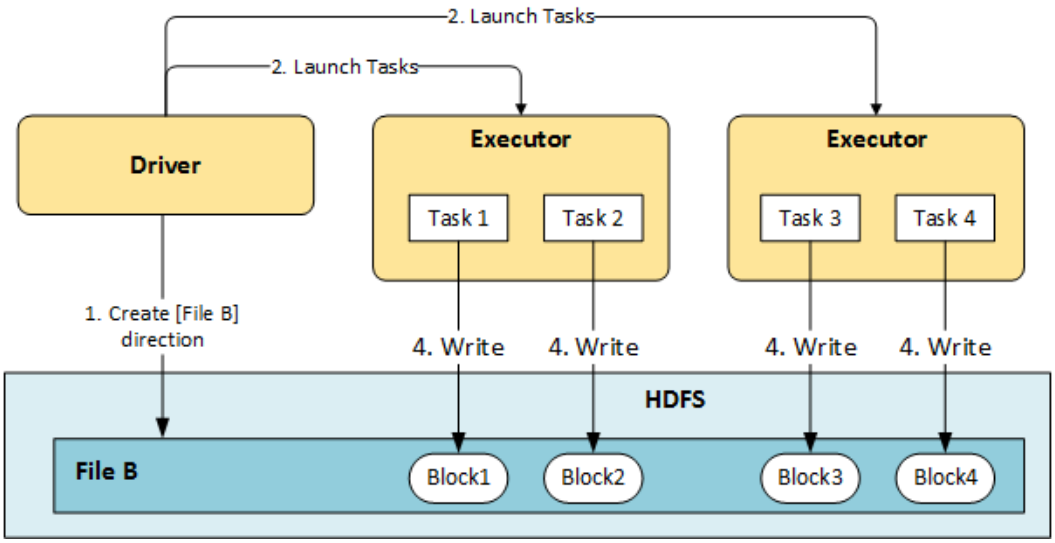


读取文件步骤的详细描述如下所示：

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2-43所示。

图 2-43 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。

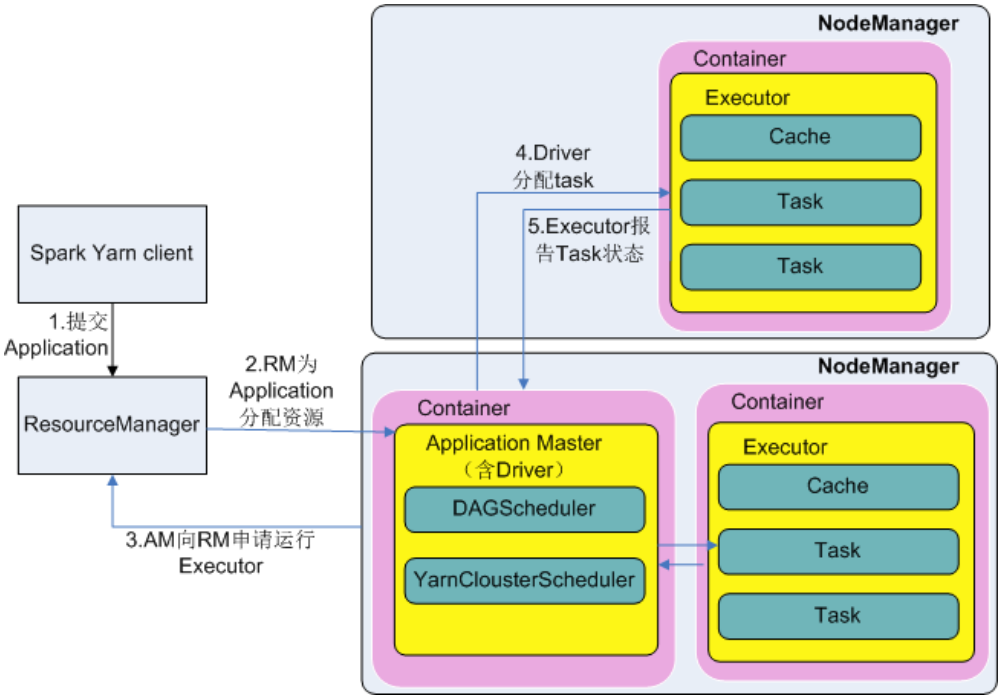
Spark 和 YARN 的配合关系

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2-44所示。

  图 2-44 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。

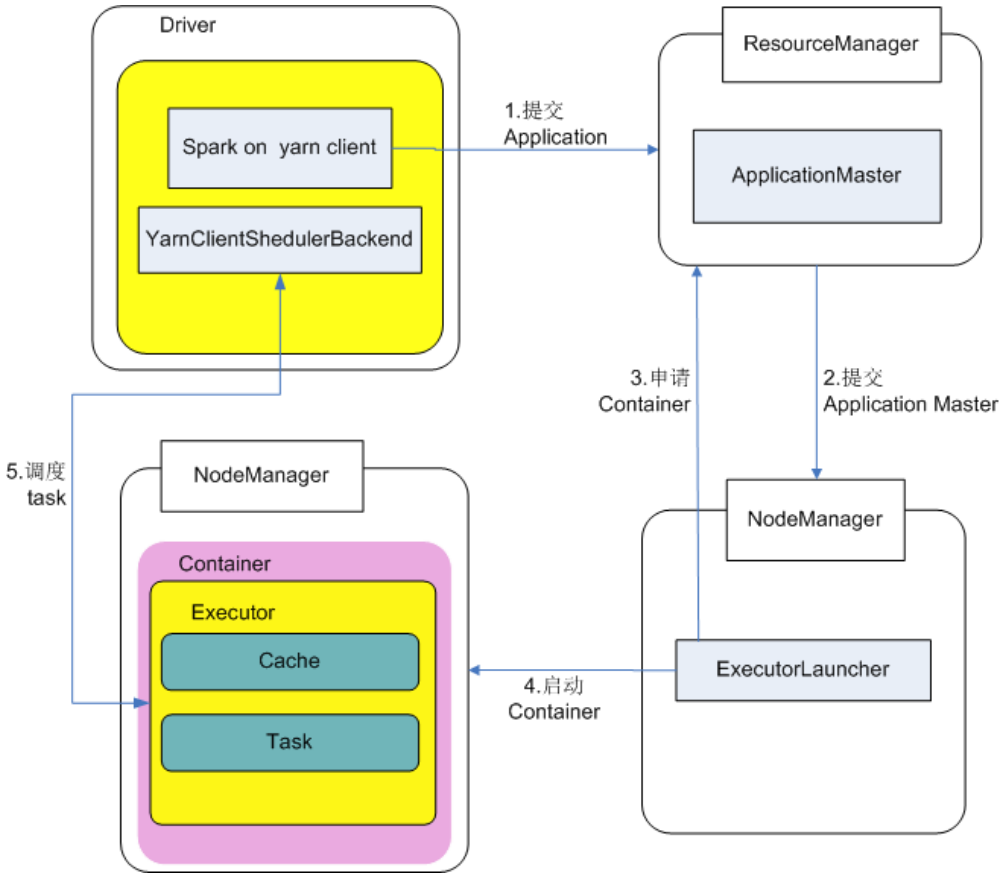
  d. Driver分配Task给Executor执行。

  e. Executor执行Task并向Driver汇报运行状况。

- YARN Client模式

  运行框架如图2-45所示。

图 2-45 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。

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。

2.3.9 ZooKeeper

2.3.9.1 基本原理

简介

ZooKeeper是一个分布式、高可用性的协调服务。在华为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。

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2-46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 2-46 ZooKeeper 结构

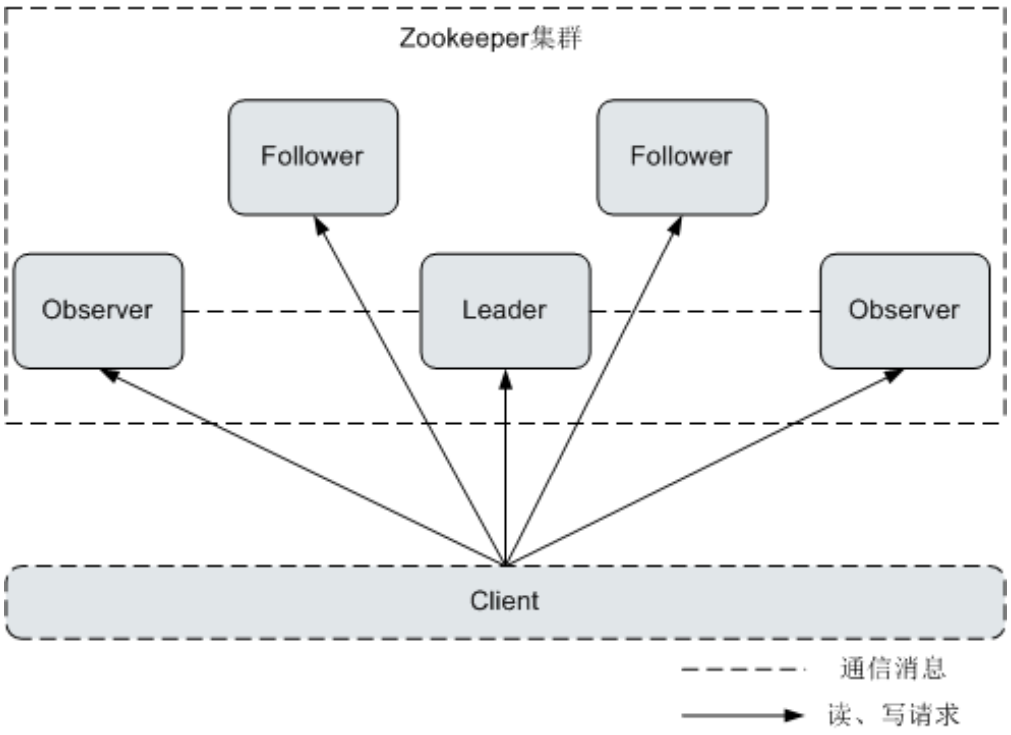


图2-46中各部分的功能说明如表2-11所示。

表 2-11 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领导者，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：需要从管理员处获取一个“人机”用户，用于登录FusionInsight HD平台并通过认证，并且获取到该用户的keytab文件。
- 票据方式：从管理员处获取一个“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

  - 获取用户和相应keytab文件的方法，请参见《管理员指南》。
  - 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。如果需要延长该用户keytab的有效期，请参见《管理员指南》修改用户的密码策略并重新获取keytab。
  - 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。

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**

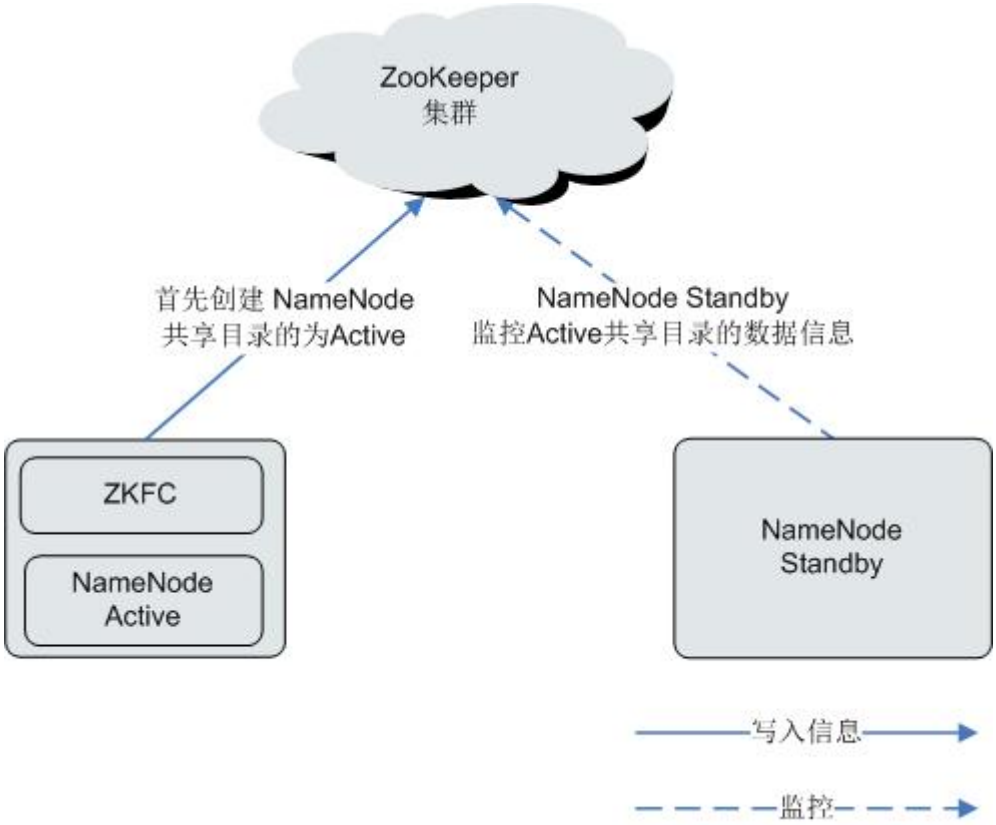
  客户端直接向Leader、Follower或Observer读取数据。

2.3.9.2 与组件的关系

ZooKeeper 和 HDFS 的配合关系

ZooKeeper与HDFS的关系如图2-47所示。

图 2-47 ZooKeeper 和 HDFS 的关系



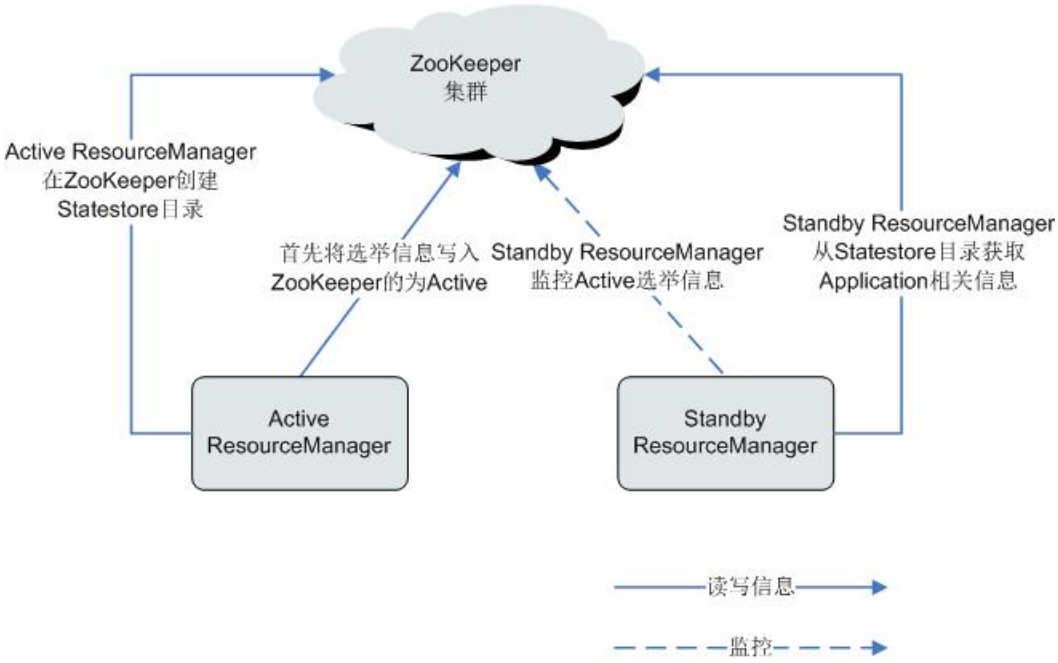
ZKFC（ZKFailoverController）作为一个ZooKeeper集群的客户端，用来监控NameNode的状态信息。ZKFC进程仅在部署了NameNode的节点中存在。HDFS NameNode的Active和Standby节点均部署有zkfc进程。

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 的配合关系

ZooKeeper与YARN的关系如图2-48所示。

图 2-48 ZooKeeper 与 YARN 的关系

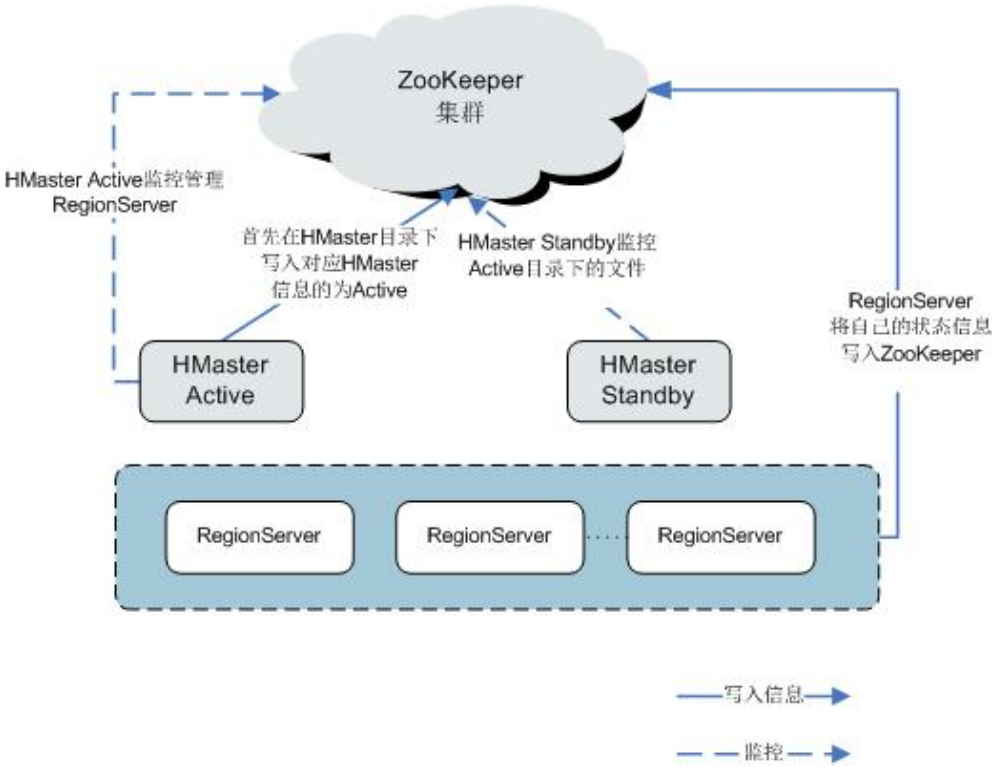


1. 在系统启动时，ResourceManager会尝试把选举信息写入ZooKeeper，第一个成功把写入ZooKeeper的ResourceManager被选举为Active ResourceManager，另一个为Standby ResourceManager。Standby ResourceManager定时去ZooKeeper监控Active ResourceManager选举信息。
2. Active ResourceManager还会在ZooKeeper中创建Statestore目录，存储Application相关信息。当Active ResourceManager产生故障时，Standby ResourceManager会从Statestore目录获取Application相关信息，恢复数据。

ZooKeeper 和 HBase 的配合关系

ZooKeeper与HBase的关系如图2-49所示。

图 2-49 ZooKeeper 和 HBase 的关系

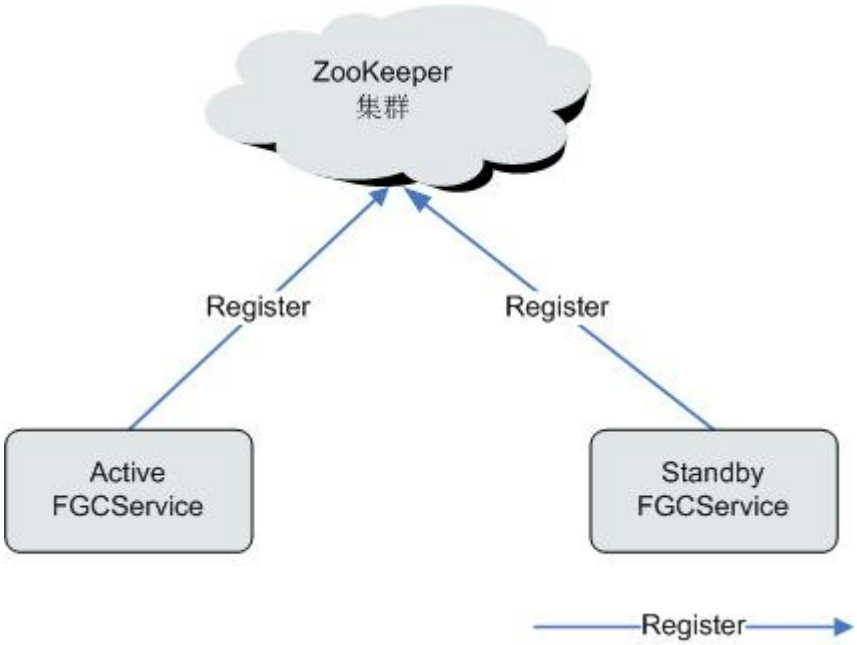


1. HRegionServer把自己以Ephemeral方式注册到ZooKeeper中。其中ZooKeeper存储HBase的如下信息：HBase元数据、HMaster地址。
2. HMaster通过ZooKeeper随时感知各个HRegionServer的健康状况，以便进行控制管理。
3. HBase也可以部署多个HMaster，类似HDFS NameNode，当HMaster主节点出现故障时，HMaster备用节点会通过ZooKeeper获取主HMaster存储的整个HBase集群状态信息。即通过ZooKeeper实现避免HBase单点故障问题的问题。

SmallFS 和 Zookeeper 的配合关系

FGCService的部署模式为HA模式。HA（High Availability）模式目的是防止单节点故障导致服务不可用。为了支持HA模式，FGCService依赖于ZooKeeper。

图 2-50 ZooKeeper 和 SmallFS 的关系



2.3.10 Hive

2.3.10.1 基本原理

简介

Hive是建立在Hadoop上的数据仓库框架，提供类似SQL的Hive Query Language语言操作结构化数据，其基本原理是将HQL语言自动转换成MapReduce任务或Spark任务，从而完成对Hadoop集群中存储的海量数据进行查询和分析。

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。

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2-51为Hive的结构概图。

图 2-51 Hive 结构

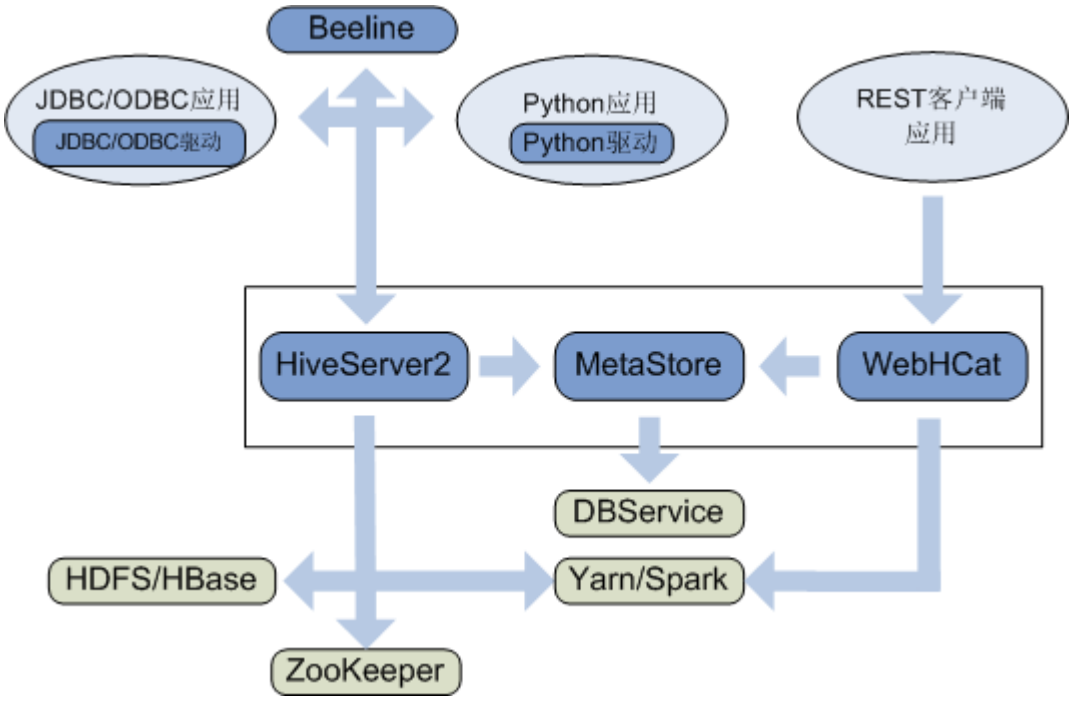
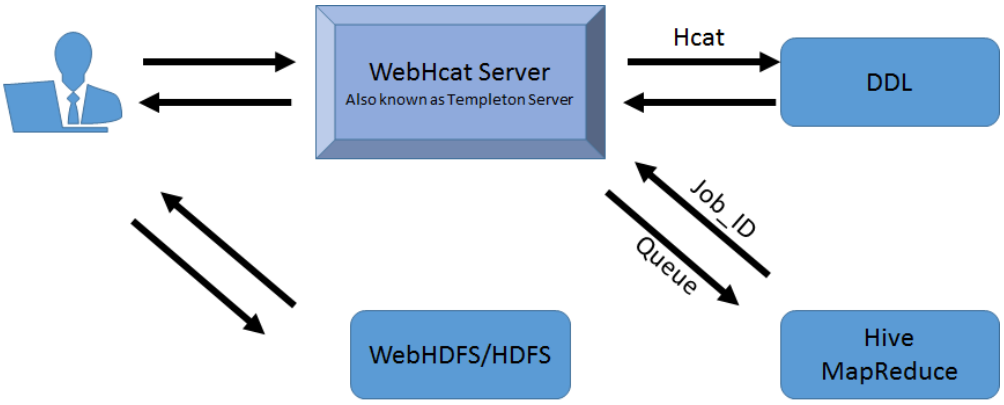


表 2-12 模块说明

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、Spark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，HiveServer的主要功能是将HQL语句转换成MapReduce任务，从而完成对海量数据的处理。
Spark集群	HiveServer将HQL语句转换成Spark任务，运行在Spark集群上。

HCatalog是Hadoop上的一个表和存储的管理层。使用不同数据处理工具（如MapReduce）的用户，使用HCatalog能够在集群上更容易的读写数据。如图2-52所示，开发人员的应用程序通过HTTP请求来访问Hadoop MapReduce（YARN）、Hive、HCatalog DDL。如果请求是HCatalog DDL命令，直接被执行；如果是MapReduce、Pig、Hive任务，则会被放置在WebHCat（Templeton开普敦）服务器的队列中，使其能够被监控进度或被停止。开发人员指定MapReduce、Pig、Hive任务的处理结果在HDFS的具体存放路径。

图 2-52 WebHCat 的逻辑架构图



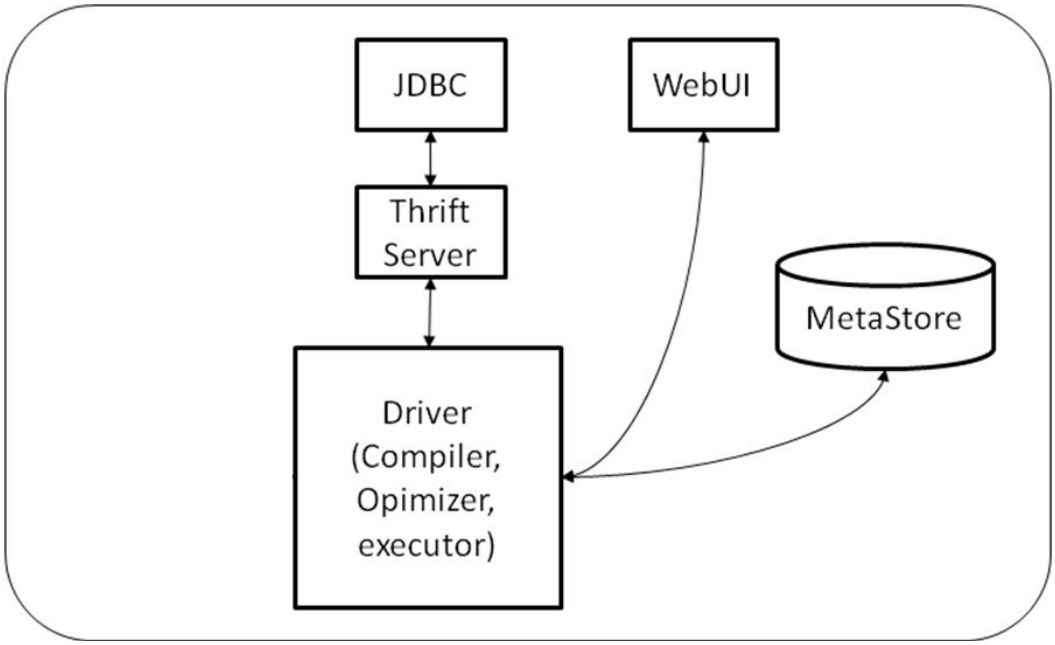
原理

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。

图2-53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HiveQL执行的生命周期并贯穿Hive任务整个执行期间。
- **Compiler** - 编译HiveQL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HiveQL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和ODBC的服务端，并将Hive和其他应用程序集成起来。
- **Clients** - 包含Web UI和JDBC/ODBC接口，为用户访问提供接口。

图 2-53 Hive 结构



2.3.10.2 与组件的关系

Hive 与 HDFS 间的关系

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 间的关系

Hive所有的数据计算都依赖于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 DBService 间的关系

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，这些信息需要存放在一个关系型数据库中，由MetaStore维护和处理。在FusionInsight HD产品中，这个关系型数据库由DBService组件维护。

2.3.11 FTP-Server

2.3.11.1 基本原理

简介

FTP-Server是一个纯Java的、基于现有开放的FTP协议的FTP服务。FTP-Server支持FTP、FTPS协议，每个服务都支持PORT、PASSIVE数据通信协议。用户或业务组件可

通过通用的FTP客户端、FTP协议对HDFS文件系统进行基本的操作，如：文件上传，文件下载，目录查看，目录创建，目录删除，文件权限修改等。

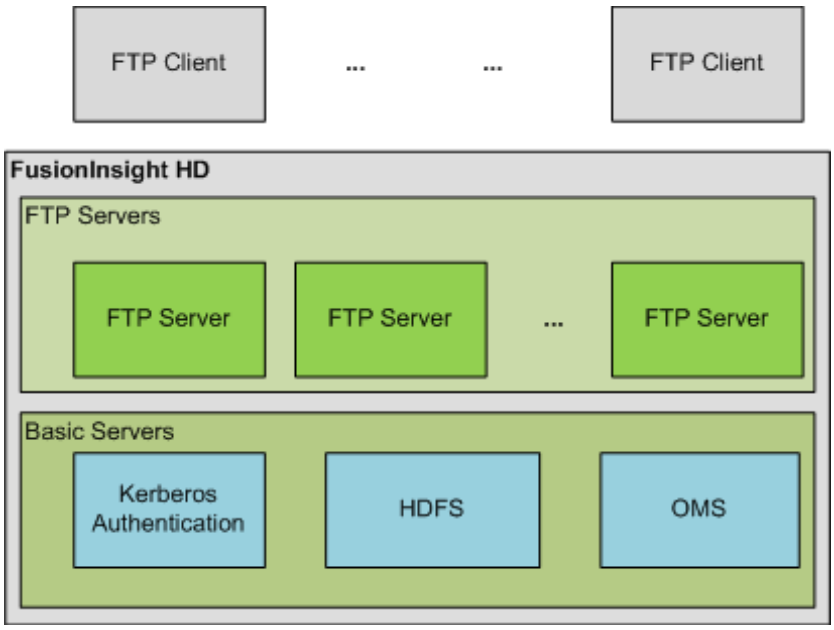
- 支持FTPS协议，FTPS是基于SSL保护的FTP协议，数据进行加密传输，数据传输受到保护。FTP协议存在安全风险，建议使用FTPS安全协议。
- 支持PORT、PASSIVE数据通信协议。
- 利用集群提供的kerberos认证服务进行用户鉴权认证。

结构

FTP-Server服务由多个FTP-Server进程或FTPS-Server进程组成。如图2-54所示。

FTP-Server服务可以部署在多个节点上，每个节点上只有一个FTP-Server实例，每个实例只有一个FTP Server进程。

图 2-54 FTP-Server 结构



FTP客户端（FTP Client）

FTP客户端用于接入FTP服务端，对数据进行上传、下载等操作。FTP客户端集成于业务应用中。

FTP服务端（FTP Servers）

FTP服务端对外提供标准的FTP接口，用于支持FTP客户端访问HDFS文件系统，其提供了大部分FTP协议命令。

FTP服务端底层服务基于FusionInsight HD的基本服务，用户管理基于Kerberos安全认证服务，数据存储基于HDFS服务，服务配置基于OMS服务。

基础服务（Basic Servers）

FTP服务端使用的基础服务有三个，Kerberos安全服务，HDFS服务和OMS服务。

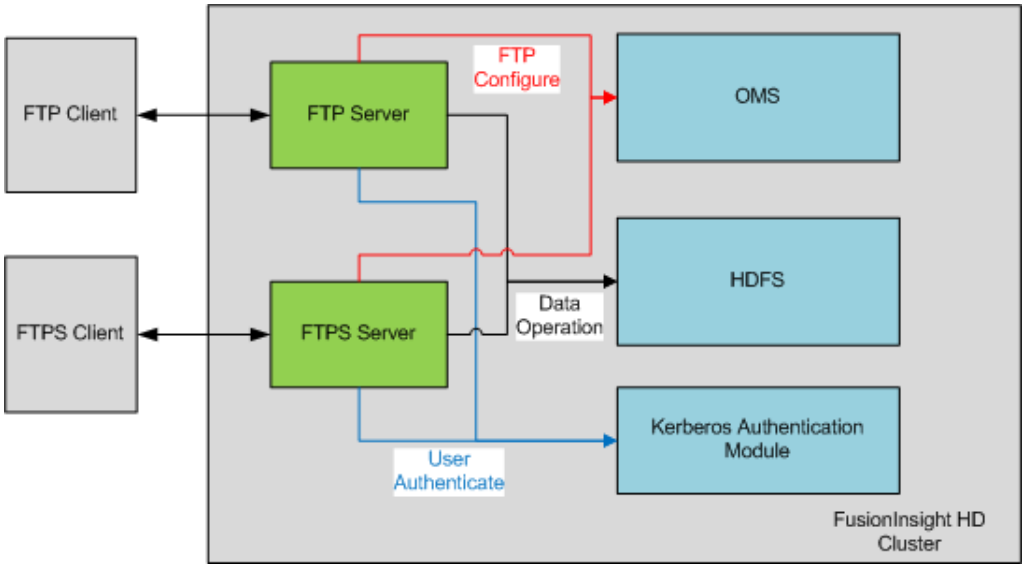
- Kerberos安全服务主要用于支持FTP用户管理和用户登录。

- HDFS服务主要用于数据存储。
- OMS服务主要用于FTP服务的参数配置和服务的启停。

原理

FTP-Server的数据方位流程介绍如图2-55所示。

图 2-55 FTP-Server 数据访问流程



1. FTP客户端使用FTP服务的IP、端口号，连接到FTP服务。
2. FTP服务端使用这些信息到Kerberos模块做用户认证。
3. 认证通过后，FTP服务端访问HDFS，将文件信息返回给客户端。
4. 客户端使用标准的FTP协议，上传、下载文件，管理HDFS文件目录。

安全

FTP协议通讯未做任何加密，通讯内容、用户的用户名、密码、传输数据容易被窃取，因此在非可信的网络，推荐使用FTPS协议通讯。FusionInsight HD提供的FTP-Server为支持企业、金融基本的应用，提供了FTPS加密协议，传输时对数据进行加密传输，有效防止信息泄漏。客户端使用FTPS协议进行连接时，只支持隐式的FTP over TLS加密方式。

FTP-Server默认关闭FTP协议的FTP-Server进程，管理员可通过FTP服务配置界面进行配置，重启服务后才能连接（使用业务IP连接）。

每个节点默认支持16个FTP/FTPS（用户或客户端）连接。若对性能有要求，建议采用FTPS协议，命令通道加密、数据通道不加密的方式。

2.3.11.2 与组件的关系

FTP-Server 与 HDFS 间的关系

HDFS是FTP-Server的存储系统，用户上传的数据全部存放在HDFS对应的目录中，用户通过FTP命令对HDFS中的文件进行操作。

FTP-Server 与 Kerberos 间的关系

Kerberos Authentication Module是FTP-Server的鉴权模块，FTP-Client在进行连接时需要将用户名和密码发送给FTP-Server，FTP-Server接收到用户名和密码之后利用Kerberos服务进行鉴权认证，检查该用户的密码是否正确并检查该用户是否有权限连接FTP-Server。

2.3.12 Loader

2.3.12.1 基本原理

简介

Loader基于开源Sqoop组件1.99.x版本进行了功能增强，实现FusionInsight HD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FusionInsight HD的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。

结构

Loader模型主要由Loader Client和Loader Server组成，如图2-56所示。

图 2-56 Loader 框架架构

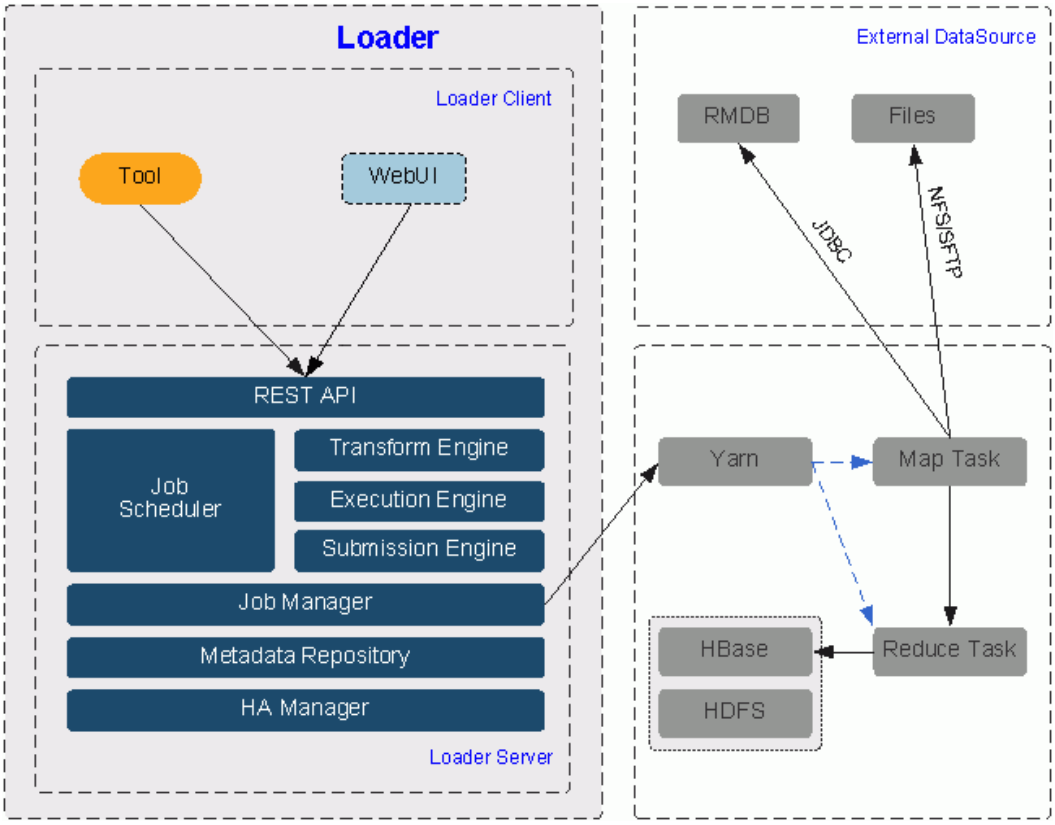


图2-56中各部分的功能说明如表2-13所示。

表 2-13 结构图说明

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。

原理

通过MapReduce实现并行执行和容错

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

1. 在MapReduce作业的Map阶段中从外部数据源抽取数据。
2. 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。
3. 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

1. 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）。
2. 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

1. 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。
2. 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

1. 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。
2. 在MapReduce作业的提交阶段，将文件从临时目录迁移到正式目录中。

2.3.12.2 与组件的关系

与Loader有交互关系的组件有HDFS、HBase、Mapreduce和Zookeeper。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MR客户端程序，完成一些数据导入导出任务。

2.3.13 Flume

2.3.13.1 基本功能

简介

Flume是一个分布式、可靠和高可用的海量日志聚合系统，支持在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，并写入各种数据接受方（可定制）的能力。Flume有如下几个特点：

- 收集、聚合事件流数据的分布式框架
- 通常用于log数据
- 采用ad-hoc方案（多跳，无中心控制节点方案）
- 声明式配置，可以动态更新配置
- 提供上下文路由功能
- 支持负载均衡和故障转移
- 完全的可扩展

结构

Flume Agent由Source、Channel、Sink组成，如图2-57所示，模块说明如表2-14所示。

图 2-57 Flume 结构图 1

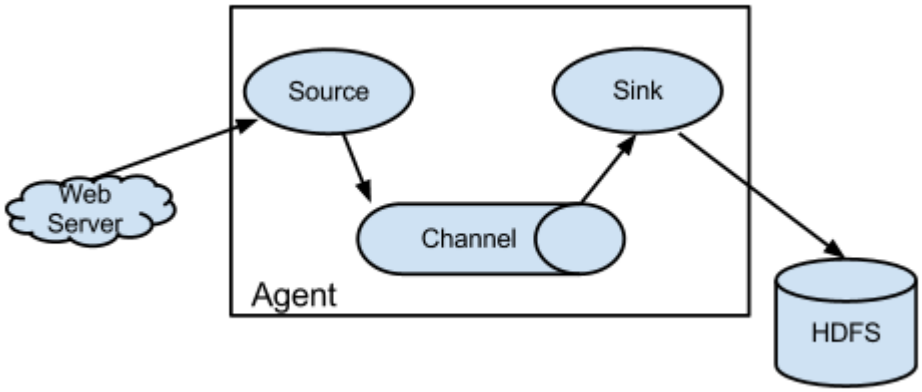
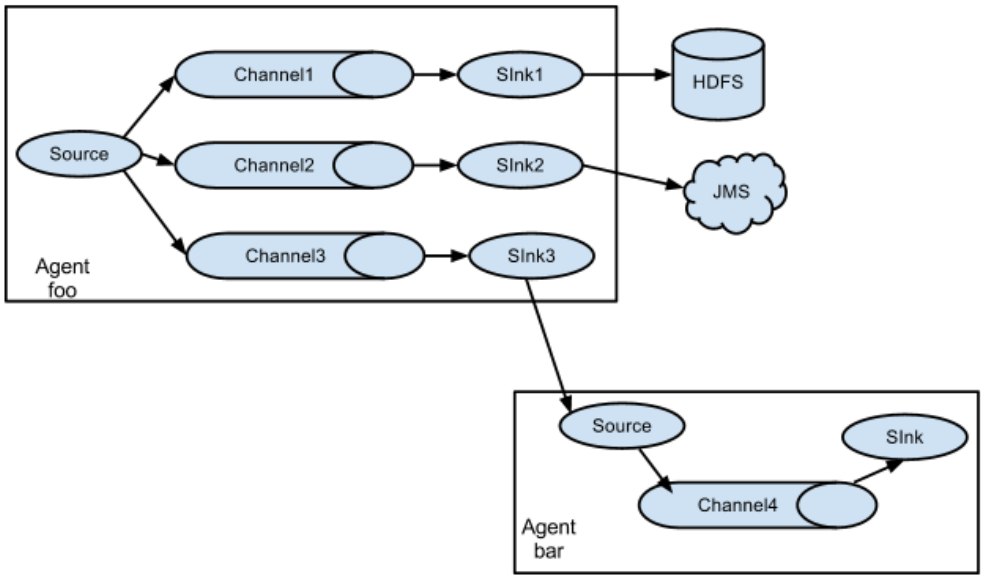


表 2-14 模块说明

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。有数据驱动和轮询2种类型的Source。 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。
Channel	Channel位于Source和Sink之间，用于缓存进来的数据，当Sink成功将数据发送到下一跳的Channel或最终目的，数据从Channel移除。 不同的Channel提供的持久化水平也是不一样的： ● 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的终端Sink，比如：HDFS、 HBase ● 自动消耗的Sinks，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。

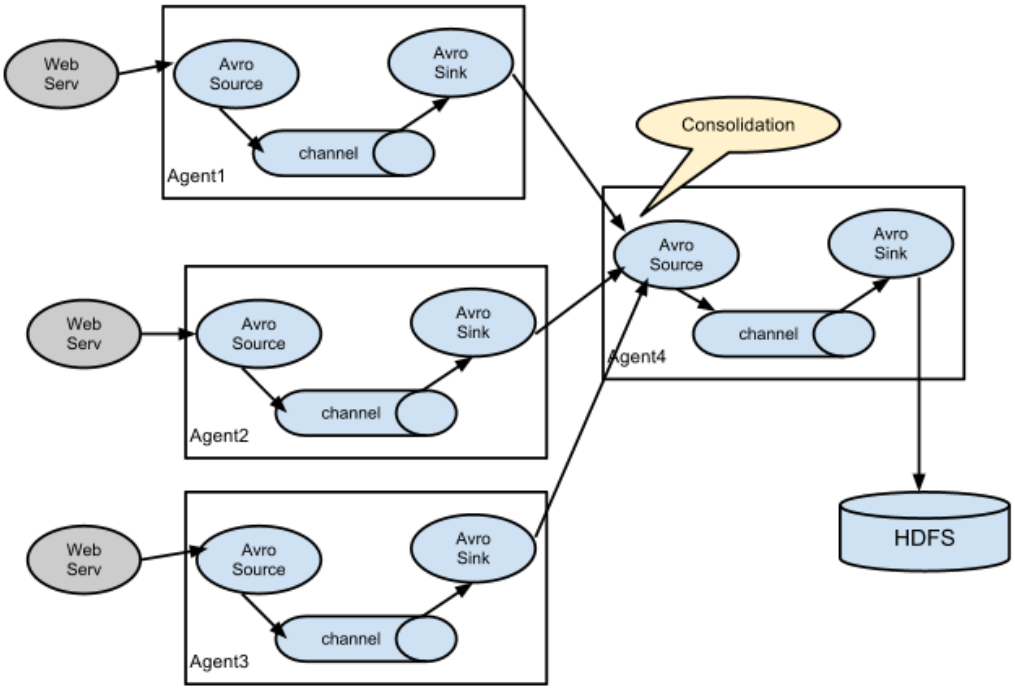
Flume也可以配置成多个Source、Channel、Sink，如图2-58所示：

图 2-58 Flume 结构图 2



Flume还支持多个Flume Agent级联，如图2-59所示。

图 2-59 Flume 级联结构图

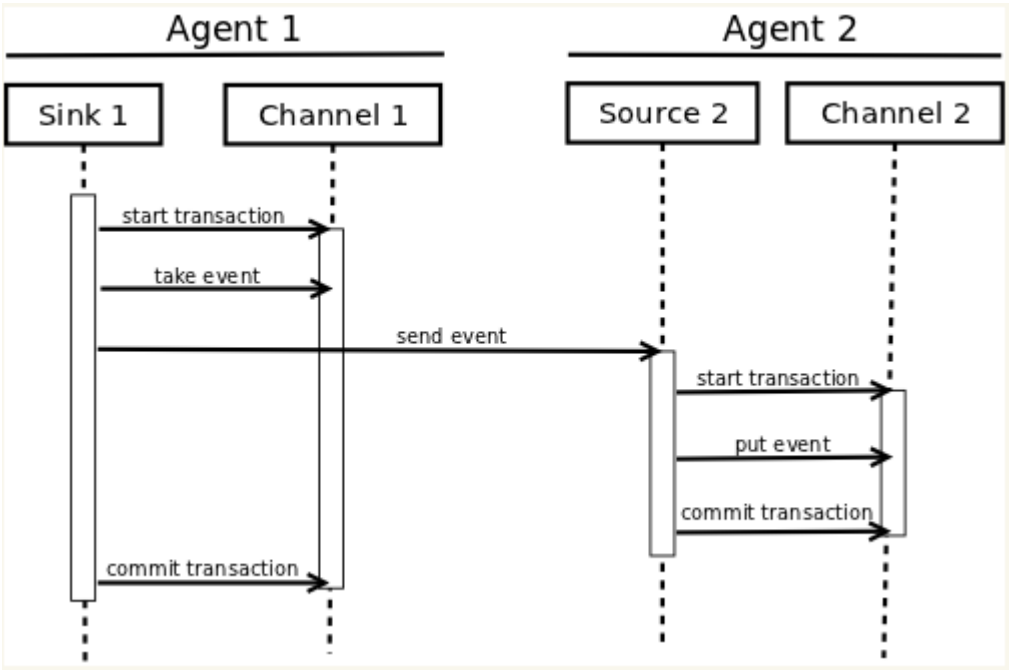


原理

Agent之间的可靠性

Agent之间数据交换流程如图2-60所示。

图 2-60 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。如果接受消息的Agent成功地接受并处理消息，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接受到发送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Transcriptions，并将上次发送失败的数据重新传输。因为commit操作已经将Transcriptions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Transcriptions。

2.3.13.2 与组件的关系

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将传输的数据全部安装配置写入HDFS中。

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将传输的数据全部安装配置写入HBase中。

2.3.14 Metadata

2.3.14.1 基本原理

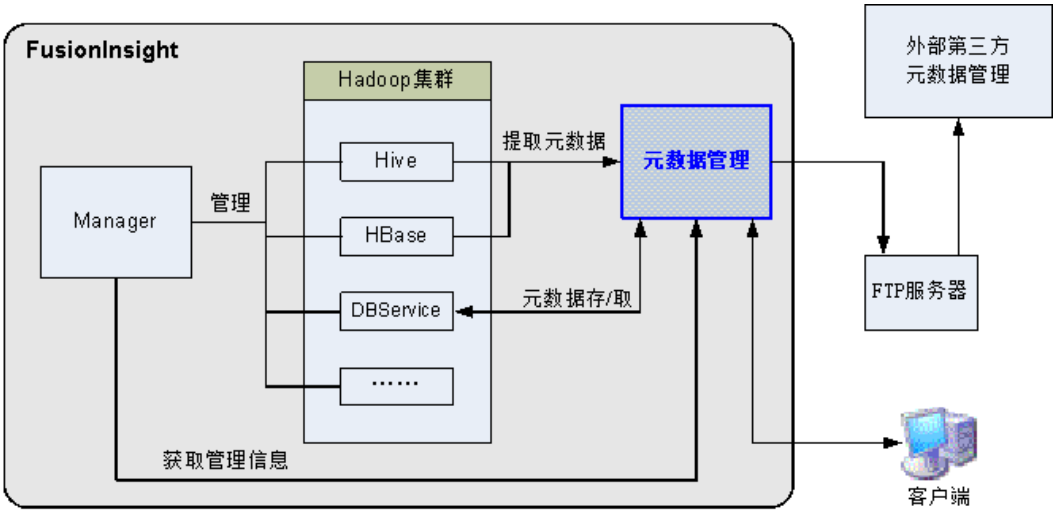
简介

MetaData（简称MDM），全称为元数据管理，可为FusionInsight HD数据仓库类型的组件（Hive和HBase）提供元数据的抽取能力，并且可以由人工为每个元数据进行标签设定，用于后向的数据分析、搜索等扩展功能。

原理

MDM可对FusionInsight HD系统中的Hive以及HBase数据库进行元数据抽取以及对外转储。MDM通过FusionInsight HD框架的安装过程，获取到Hive以及HBase的连接方式并有效接入认证，从而获取到这两个数据库的元数据。

图 2-61 元数据逻辑架构图



元数据管理的原理说明如下：

1. MDM从Manager处获取Hadoop集群的基本信息，例如HBase的RegionServer的节点部署信息、保存了Hive元数据的DBService信息等。
2. 根据1获取的信息，MDM从Hive以及HBase中提取元数据，并将元数据信息保存在DBService上。用户可通过客户端登录FusionInsight HD系统，查看元数据信息。
3. 将提取出来的元数据通过外部FTP服务器上传到第三方元数据管理系统中，以支撑更高层级的元数据管理。

2.3.14.2 与组件的关系

HBase 和 MetaData 的关系

MDM通过在FusionInsight Manager框架的安装过程，获取到Hbase的连接方式并有效接入认证，从而获取到HBase数据库的元数据。

Hive 和 MetaData 的关系

MDM通过在FusionInsight Manager框架的安装过程，获取到Hive的连接方式并有效接入认证，从而获取到Hive数据库的元数据。

DBService 和 MetaData 的关系

MetaData把从Hive和HBase处获得的元数据存放在DBService上，并通过DBService提供这些元数据的备份与恢复功能，还可将元数据通过外部FTP服务器提取到外部系统。

2.3.15 Hue

2.3.15.1 基本原理

简介

Hue提供了FusionInsight HD应用的图形化用户界面。Hue支持展示多种组件，目前支持HDFS，MapReduce，Hive以及Solr。通过Hue可以在界面针对组件进行以下操作。

- HDFS：创建文件/目录，修改文件/目录权限，上传、下载文件，查看、修改文件等操作。
- 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。
- Hive：编辑、执行HQL，也可通过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HiveSQL，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21及以上版本。
- Solr：支持基于Solr进行搜索的应用，并提供可视化的数据视图。

结构

Hue是建立在Django Python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。（Django Python是开放源代码的Web应用框架。）

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2-62所示。

图 2-62 Hue 架构示意图

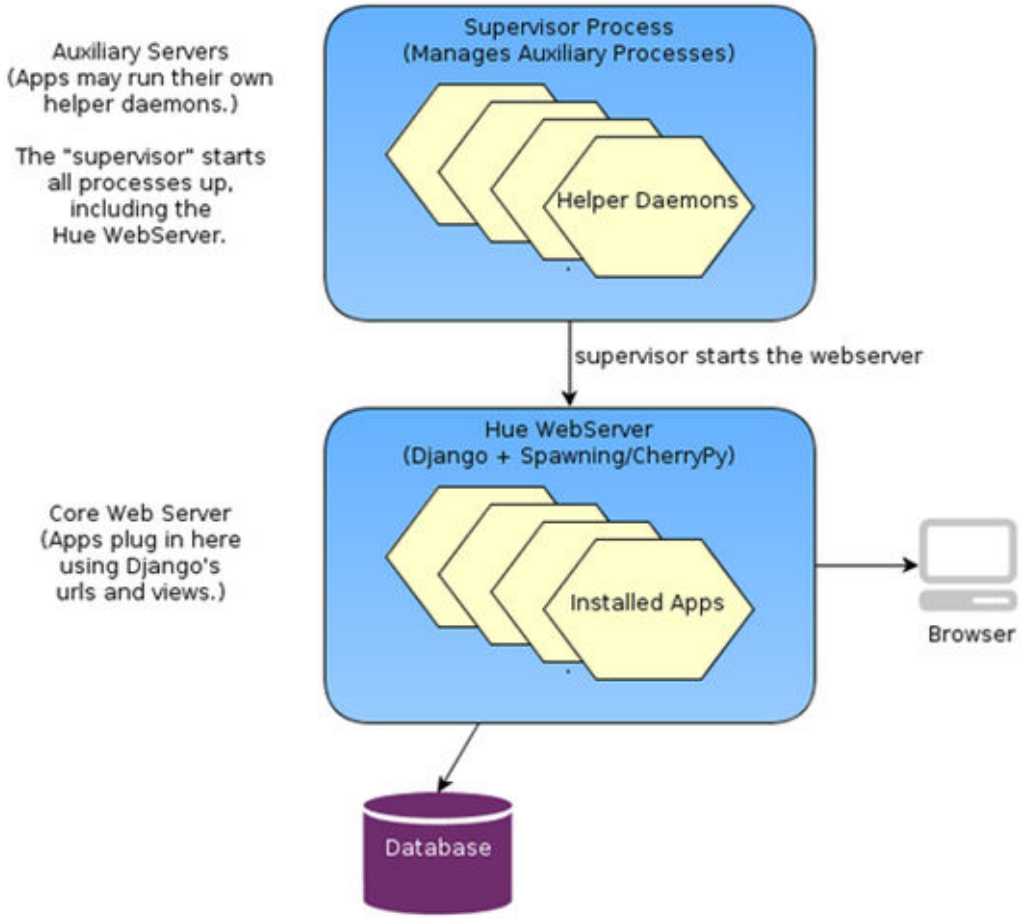


图2-62中各部分的功能说明如表2-15所示。

表 2-15 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APPs的持久化数据。

2.3.15.2 与组件的关系

Hue与Hadoop集群的交互关系如图2-63所示。

图 2-63 Hue 与 Hadoop 集群

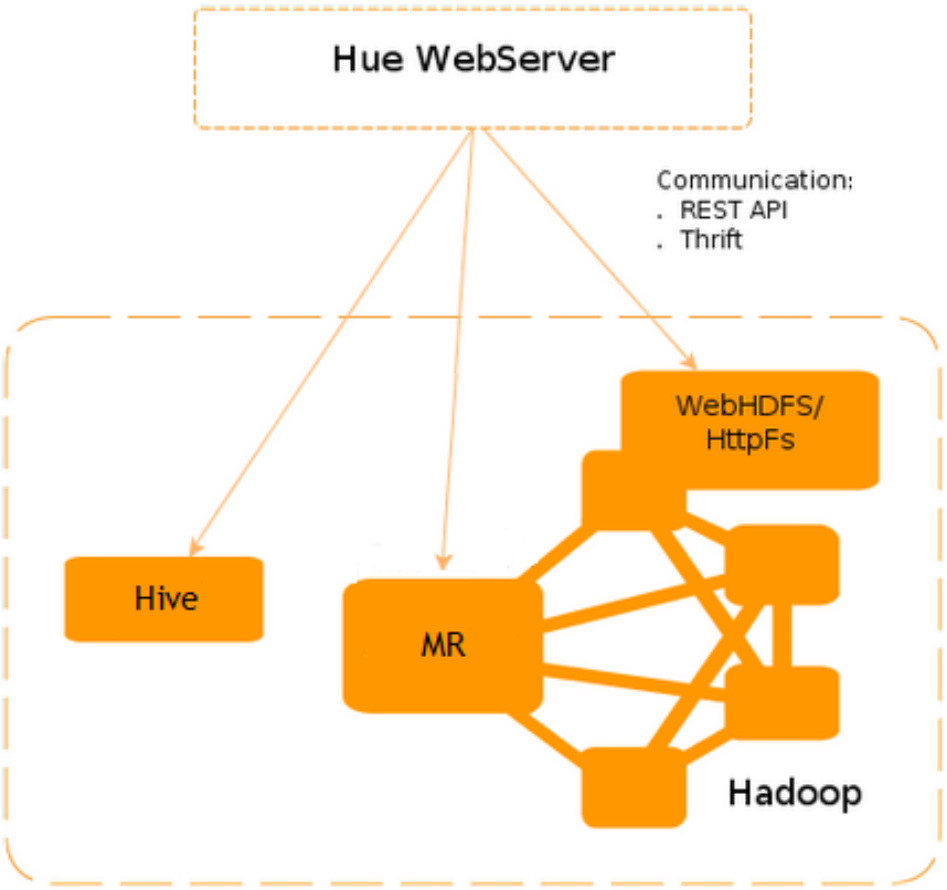


表 2-16 Hue 与 Hadoop 集群交互关系

名称	描述
Hive	Hive提供REST和THRIFT接口与Hue交互
MR(MR1/MR2-YARN)	MapReduce提供REST和THRIFT接口与Hue交互
WebHDFS/httpFs	HDFS提供REST和THRIFT接口与Hue交互
其他未标明部分	Hue支持的其他Hadoop组件，例如：Solr等，Solr提供REST接口与Hue交互

Hue 与 HDFS 交互原理

在Hue把用户请求从用户界面组装成接口数据，通过调用REST和THRIFT接口调用HDFS，通过浏览器返回结果呈现给用户。

Hue 与 Hive 交互原理

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。

Hue 与 MapReduce 交互原理

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。

Hue 与 Solr 交互原理

在Hue前端通过浏览器点击、输入等方式输入筛选参数，Hue前端将筛选参数传到后台，Hue后台调用Solr的REST接口，然后以JSON格式将Solr返回的结果返回给Hue前台，Hue前台利用图标控件将结果展示出来。

2.3.16 Solr

2.3.16.1 基本原理

简介

Solr是一个高性能、基于Lucene的全文检索服务器。Solr对Lucene进行了扩展，提供了比Lucene更为丰富的查询语言，并实现了强大的全文检索功能、高亮显示、动态集群，具有高度的可扩展性。同时从Solr 4.0版本开始，支持SolrCloud模式，该模式下能够进行集中式的配置信息、近实时搜索、自动容错等功能：

- 利用ZooKeeper作为协同服务，启动时可以指定把Solr的相关配置文件上传ZooKeeper，多机器共用。这些ZooKeeper中的配置不会再拿到本地缓存，Solr直接读取ZooKeeper中的配置信息。配置文件的变动，所有机器都可以感知到。
- 自动容错，SolrCloud对索引（collection）进行分片（shard），并对每个分片创建多个Replica。每个Replica都可以独立对外提供服务。一个Replica出现异常不会影响整个索引搜索服务；更强大的是，它还能自动的在其它机器上把失败机器上的索引Replica重建并投入使用。
- 索引和查询时的自动负载均衡，SolrCloud索引（collection）的多个Replica可以分布在多台机器上，均衡索引和查询压力。如果索引和查询压力大，可以通过扩展机器，增加Replica来减缓压力。 因此，下面的介绍主要是围绕SolrCloud展开描述的。
- Solr索引数据存储方法有多种，利用HDFS作为其索引文件的存储系统，提供高可靠性、高性能、可伸缩、准实时的全文检索系统；存放到本地磁盘，提供了更加快速的索引和查询速度。

基本概念

- Collection：在SolrCloud集群中逻辑意义上的完整的索引。它可以被划分为一个或者多个Shard，它们使用相同的Config Set。
- Config Set：Solr Core提供服务必须的一组配置文件。包括solrconfig.xml(SolrConfigXml)和schema.xml (SchemaXml)等。

- Core：即Solr Core，一个Solr实例中包含一个或者多个Solr Core，每个Solr Core可以独立提供索引和查询功能，每个Solr Core对应一个索引或者Collection的Shard的副本(replica)。
- Shard：Collection的逻辑分片。每个Shard都包含一个或者多个replicas，通过选举确定哪个是Leader。
- Replica：Shard的拷贝。一个Replica存在于Solr的一个Core中。
- Leader：赢得选举的Shard replicas。当索引documents时，SolrCloud会传递它们到此Shard对应的leader，leader再分发它们到全部Shard的replicas。
- ZooKeeper：它在SolrCloud是必须的，提供分布式锁、处理Leader选举等功能。

结构

Solr集群方案SolrCloud由多个SolrServer进程组成，如图2-64所示，模块说明如表2-17所示。

图 2-64 Solr(SolrCloud)结构

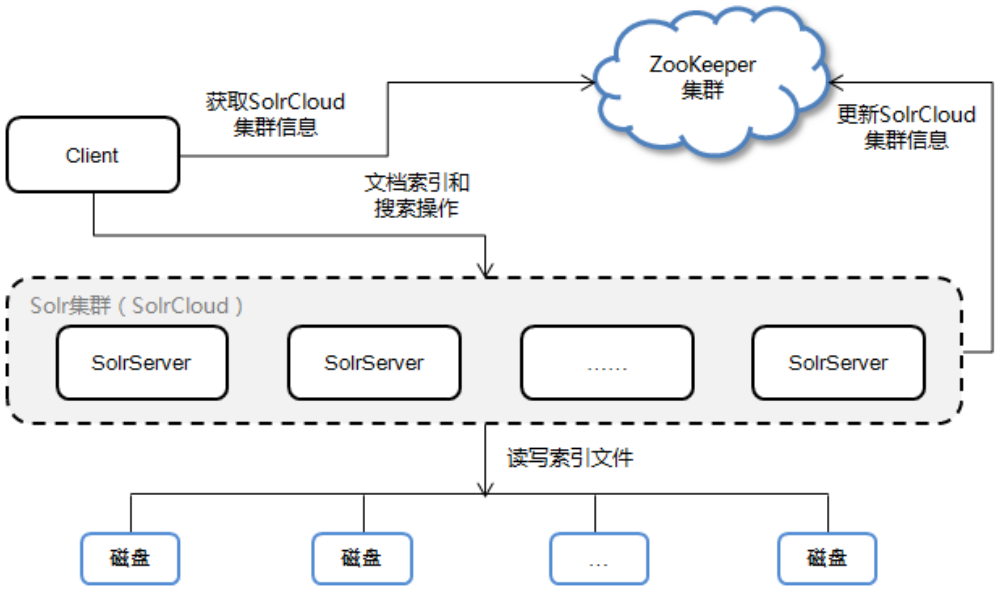


表 2-17 模块说明

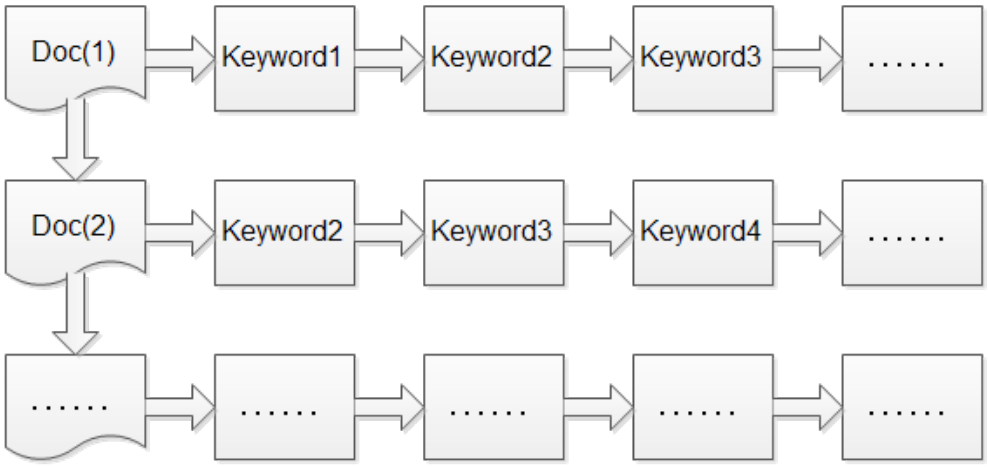
名称	说明
Client	Client使用HTTP或HTTPS协议同Solr集群（SolrCloud）中的SolrServer进行通信，进行分布式索引和分布式搜索操作。
SolrServer	SolrServer负责提供创建索引和全文检索等服务，是Solr集群中的数据计算和处理单元。
ZooKeeper集群	ZooKeeper为Solr集群中各进程提供分布式协作服务。各SolrServer将自己的信息（collection配置信息、SolrServer健康信息等）注册到ZooKeeper中，Client据此感知各个SolrServer的健康状态来决定索引和搜索请求的分发。

原理

- **倒排序索引**

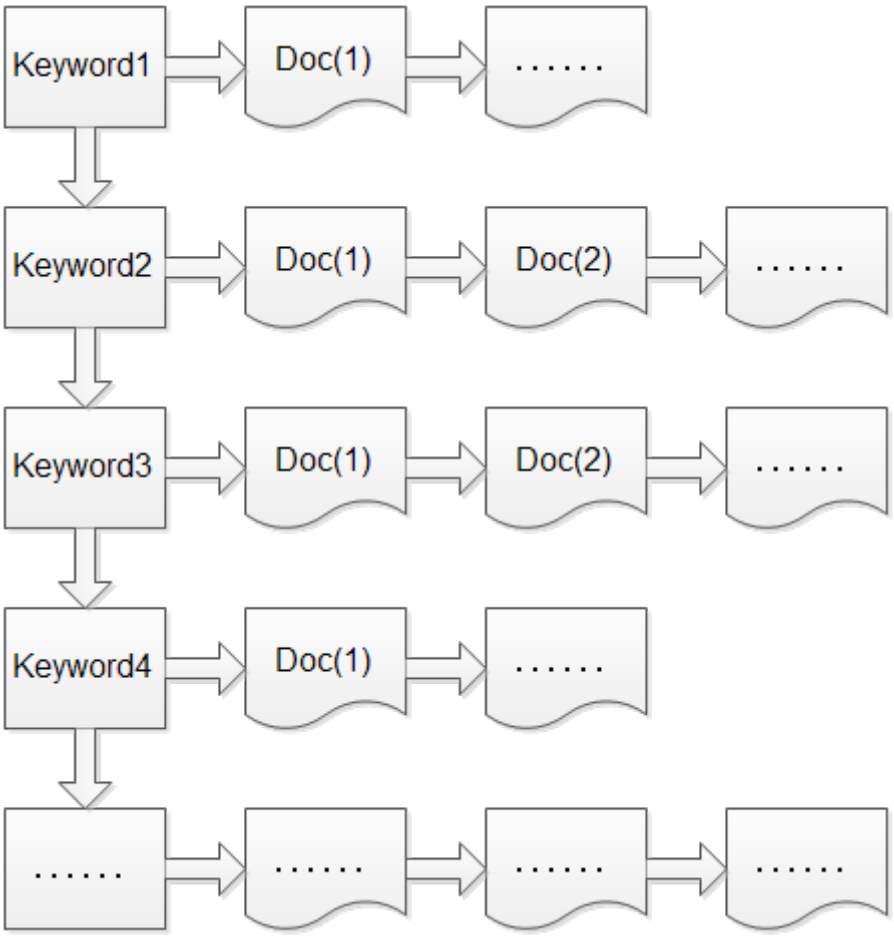
  传统的搜索方式（正排序索引，如图2-65所示）是从关键点出发，然后再通过关键点找到关键点代表的信息中能够满足搜索条件的特定信息，即通过KEY寻找VALUE。通过正排序索引进行搜索，就是从通过文档编号找关键词。

  图 2-65 正排序索引

  而Solr（Lucene）的搜索则是采用了倒排序索引（如图2-66所示）的方式，即通过VALUE找KEY。而在中文全文搜索中VALUE就是要搜索的关键词，存放所有关键词的地方叫词典。KEY是文档标号列表（通过文档标号列表可以找到出现过要搜索关键词--VALUE的文档），具体如下面的图所示：通过倒排序索引进行搜索，就是通过关键词查询相对应的文档编号，再通过文档编号找文档，类似于查字典，或通过查书目录查指定页码书的内容。

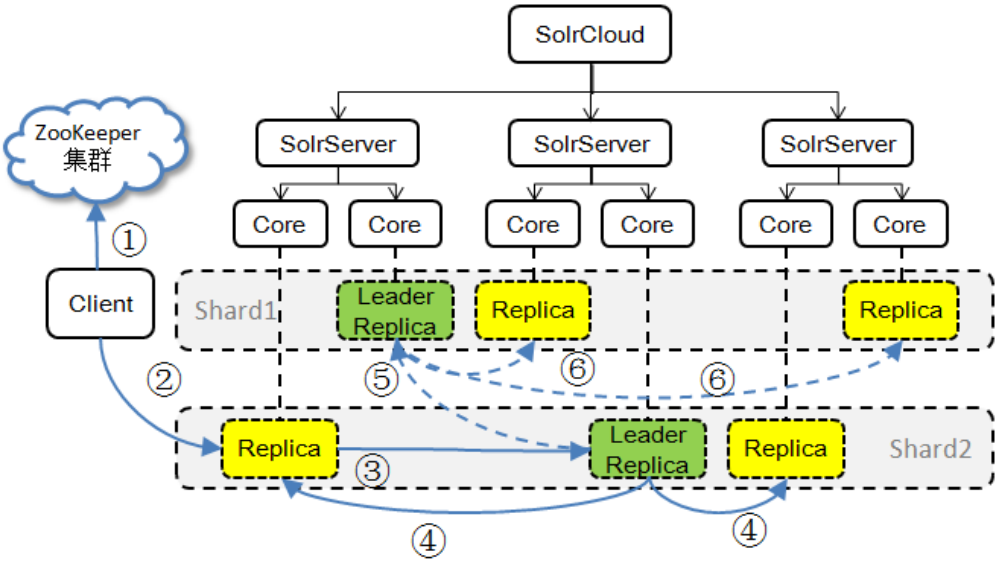
图 2-66 倒排序索引



- **分布式索引操作流程**

  Solr分布式索引操作流程如图2-67所示。

图 2-67 分布式索引操作流程



操作流程说明如下：

a. 当Client发起一次文档索引请求时，首先将从ZooKeeper集群中获取SolrCloud中SolrServer的集群信息，根据请求中的collection信息，获取任意一台包含该collection信息的SolrServer；

b. Client把文档索引请求发送给SolrServer中该collection对应shard中的一个Replica进行处理；

c. 如果该Replica不是Leader Replica，则该Replica会把文档索引请求再转发给和自己相同shard中相对应的Leader Replica；

d. 该Leader Replica在本地完成文档的索引后，会再把文档索引请求路由给本Shard中的其他Replica进行处理；

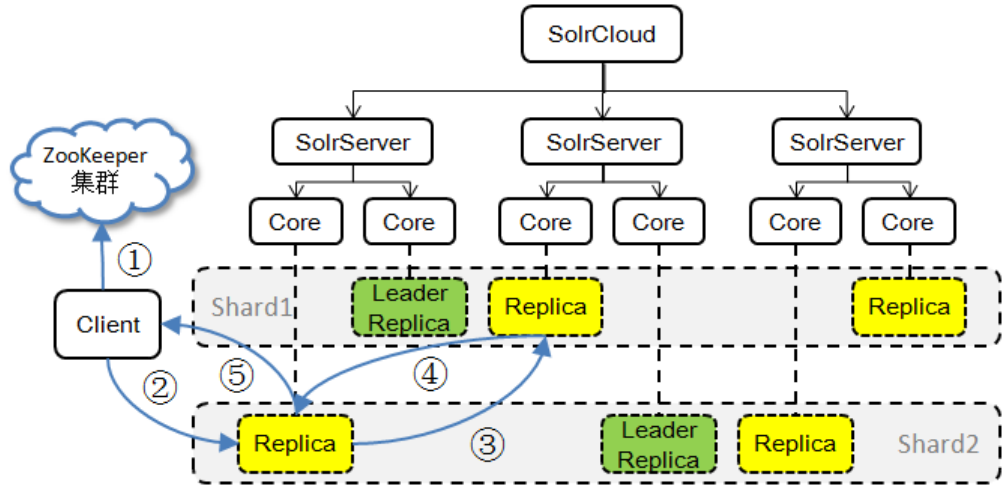
e. 如果该文档索引的目标shard并不是本次请求的Shard，那么该Shard的Leader Replica会将文档索引请求再次转发给目标Shard的Leader Replica；

f. 目标Shard的Leader Replica在本地完成文档的索引后，会再把文档索引请求再次路由给本Shard的其他Replica进行处理。

- **分布式搜索操作流程**

Solr分布式搜索操作流程如图2-68所示。

图 2-68 分布式搜索操作流程



操作流程说明如下：

a. 当Client发起一次搜索请求时；Client首先将通过ZooKeeper会获取到SolrServer服务器集群信息，并随机选取一个含有该collection的SolrServer；

b. Client把搜索请求发送到该Collection在SolrServer上相对应Shard中的任意一个Replica（可以不为Leader Replica）进行处理；

c. 该Replica再根据查询索引的方式，启动分布式查询，基于Collection的Shard个数（在图2-68中为2个，Shard1和Shard2），把查询转换为多个子查询，并把每个子查询分发到对应Shard的任意一个Replica（可以不为Leader Replica）中进行处理；

d. 每个子查询完成查询操作后，并查询结果返回；

e. 首次收到查询请求的Replica收到各个子查询的查询结果后，对各个查询结果进行合并处理，然后把最终的查询结果返回给Client。

2.3.16.2 与组件的关系

Solr 与 HDFS 间的关系

Solr是Apache基金会下的项目，也是Apache Hadoop项目生态系统中重要的一员，Solr可利用HDFS作为其索引文件存储系统。Solr位于结构化存储层，HDFS为Solr提供了高可靠性的存储支持。Solr中的所有索引数据文件都可以存储在HDFS文件系统上。

Solr 与 HBase 间的关系

HBase提供海量数据存储功能，是一种构建在HDFS上的分布式、面向列的存储系统。Solr索引HBase数据是将HBase数据写到HDFS的同时，Solr建立相应的HBase索引数据。其中索引id与HBase数据的rowkey对应，保证每条索引数据与HBase数据的唯一，实现HBase数据的全文检索。

2.3.17 Oozie

2.3.17.1 基本原理

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。

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2-69**所示。

图 2-69 框架架构

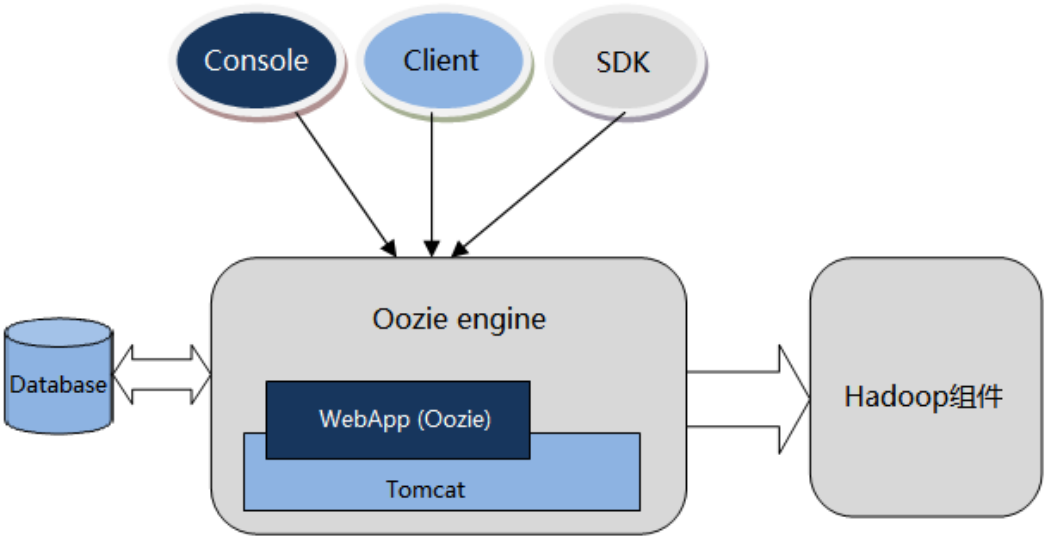


图2-69中各部分的功能说明如表2-18所示。

表 2-18 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。

原理

Oozie是一个工作流引擎服务器，用于运行HD 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MapReuce、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。

2.3.18 Kafka

2.3.18.1 基本原理

简介

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。

结构

Kafka是一个分布式、分区化、多副本的消息发布-订阅系统。生产者（Producer）将消息发布到Kafka主题（Topic），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 2-70 Kafka 结构

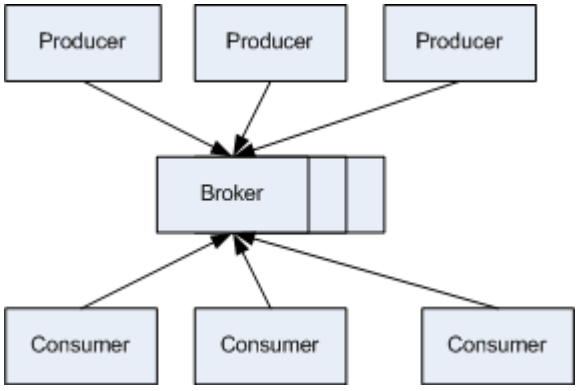
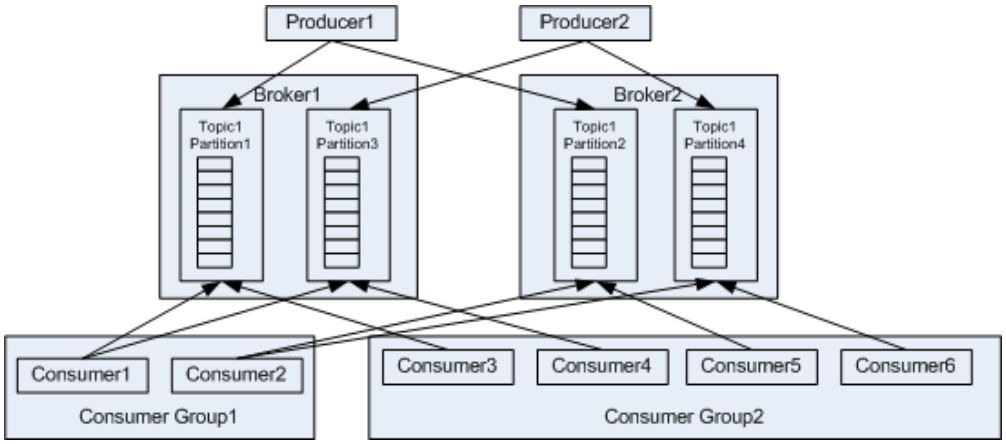


表 2-19 结构图说明

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。

各模块间关系如下：

图 2-71 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，它们就像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。

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cache，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，并及时作出调整。

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防止数据丢失。

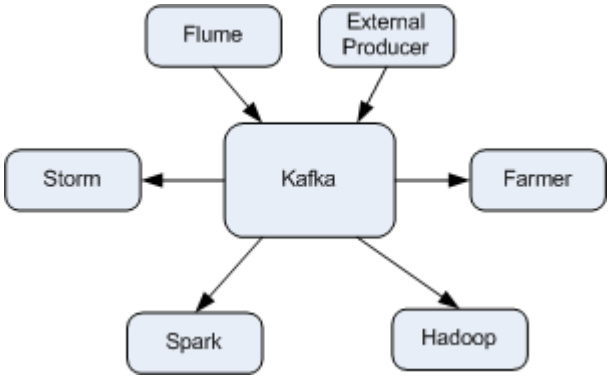
- 分布式

  分布式系统，易于向外扩展。所有的Producer、Broker和Consumer都支持部署多个形成分布式的集群。无需停机即可扩展系统。

2.3.18.2 与组件的关系

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。可以实时接受来自外部的消息，并提供给在线以及离线业务进行处理。具体的关系如下图所示：

图 2-72 与组件关系



2.3.19 Streaming

2.3.19.1 基本原理

功能

Streaming基于开源Apache Storm，是一个分布式、可靠、容错的实时计算系统。用于对大规模流式数据提供实时处理。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

Streaming以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）。相对于标准的SQL，CQL加入了窗口的概念，适用于KPI统计等业务场景。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

结构

Streaming服务由主备Nimbus进程、对应的UI进程和多个Supervisor进程组成，如图2-73所示。

图 2-73 Streaming 结构

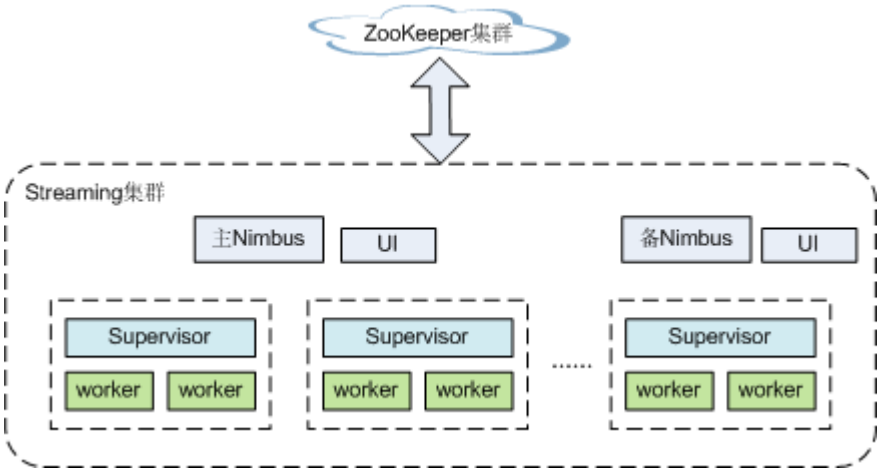


表 2-20 结构图说明

名称	说明
Nimbus	Streaming服务的控制中心节点，在HA模式下包含主用Nimbus和备用Nimbus。 ● 主用Nimbus：负责接收客户端提交的任务，并在集群中分发任务给Supervisor；同时监听状态等。 ● 备用Nimbus：当主用Nimbus故障时，备用Nimbus将取代主用Nimbus对外提供服务。
Supervisor	负责监听并接受Nimbus分配的任务，根据需要启动和停止属于自己管理的Worker进程。Worker进程是运行具体处理组件逻辑的进程。每个Worker是一个JVM进程。
UI	Streaming业务监控界面，用于查看运行的拓扑情况。
ZooKeeper	ZooKeeper为Streaming服务中各进程提供分布式协作服务。主备Nimbus、Supervisor、Worker将自己的信息注册到ZooKeeper中，Nimbus据此感知各个角色的健康状态。

原理

- **基本概念**

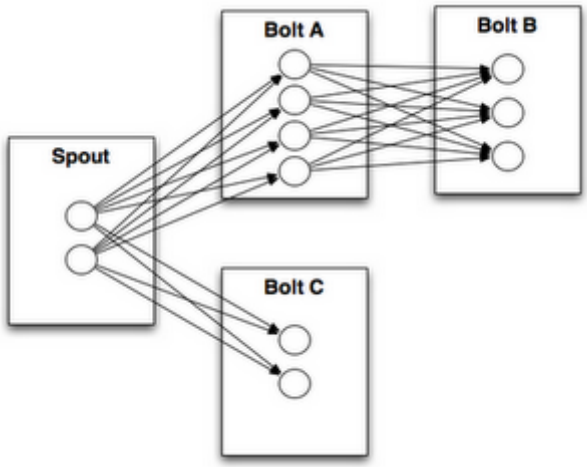
表 2-21 概念介绍

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。

概念	说明
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2-74描述了一个由Spout、Bolt组成的DAG，即Topology。图中每个矩型框代表Spout或者Bolt，矩型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 2-74 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2-22从Storm的不同部件失效的情况角度解释其容错能力：

表 2-22 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。
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它节点。
节点失效	则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。

开源特性

- 分布式实时计算框架

  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。

- 多语言支持

  除了JAVA语言，支持使用任何熟悉的语言进行逻辑开发，允许Spout或者Bolt使用标准输入和标准输出来进行消息传递，传递的消息为单行文本或者是JSON编码的多行文本。

- 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。

- 滚动升级

  支持新版本无缝升级，即软件升级时，无需重新部署业务。

- 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。

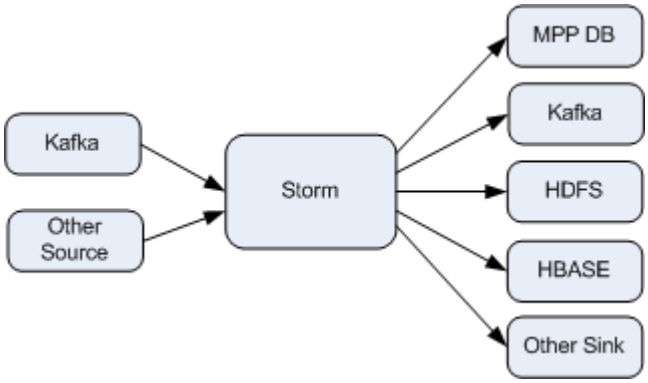
- 与外部组件集成

  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。

2.3.19.2 与组件的关系

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，在实时平台中完成高吞吐、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2-75所示：

图 2-75 组件关系图



2.3.20 Redis

2.3.20.1 基本原理

简介

Redis是一个开源的，基于网络的，高性能的key-value数据库，弥补了memcached这类key-value存储的不足，在部分场合可以对关系数据库起到很好的补充作用，满足实时的高并发需求。

Redis跟memcached类似，不过数据可以持久化，而且支持的数据类型很丰富。支持在服务器端计算集合的并，交和补集(difference)等，还支持多种排序功能。

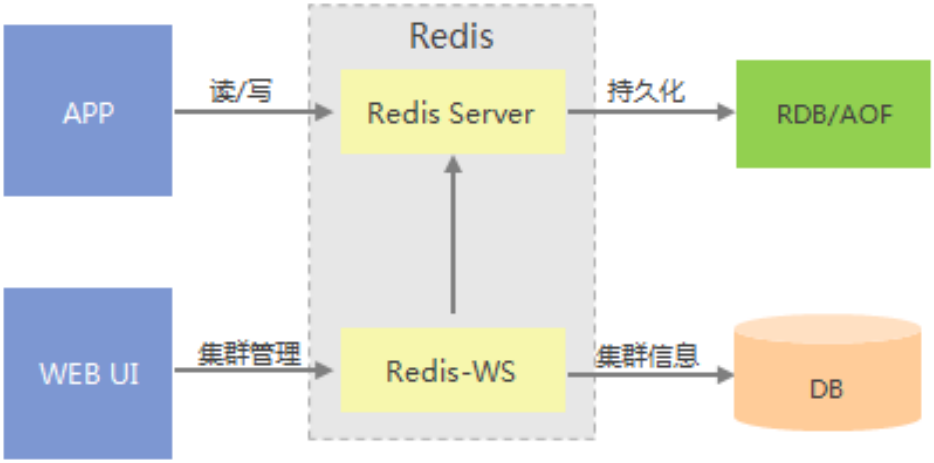
说明

Redis客户端跟服务端间的网络数据传输未加密，建议不要使用Redis存取敏感数据，否则可能存在安全风险。

结构

Redis包含Redis Server与Redis-WS，如图2-76所示。

图 2-76 Redis 逻辑架构



- Redis Server：Redis组件的核心模块，负责Redis协议的数据读写，数据持久化，主从复制，集群功能。
- Redis-WS：Redis WebService管理模块，主要负责Redis集群的创建、扩容、减容、查询、删除等操作，集群管理信息存入DB数据库。

原理

Redis持久化

Redis提供了RDB与AOF等多种不同级别的持久化方式。

- RDB持久化

  可以在指定的时间间隔内生成数据集的时间点快照（point-in-time snapshot）。

- AOF持久化

  记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF文件中的命令全部以Redis协议的格式来保存，新命令会被追加到文件的末尾。Redis还可以在后台对AOF文件进行重写，使得AOF文件的体积不会超出保存数据集状态所需的实际大小。

Redis可以同时使用AOF持久化和RDB持久化。在这种情况下，当Redis重启时，它会优先使用AOF文件来还原数据集，因为AOF文件保存的数据集通常比RDB文件所保存的数据集更完整。用户也可以关闭持久化功能，让数据只在服务器运行时存在。

Redis运行模式

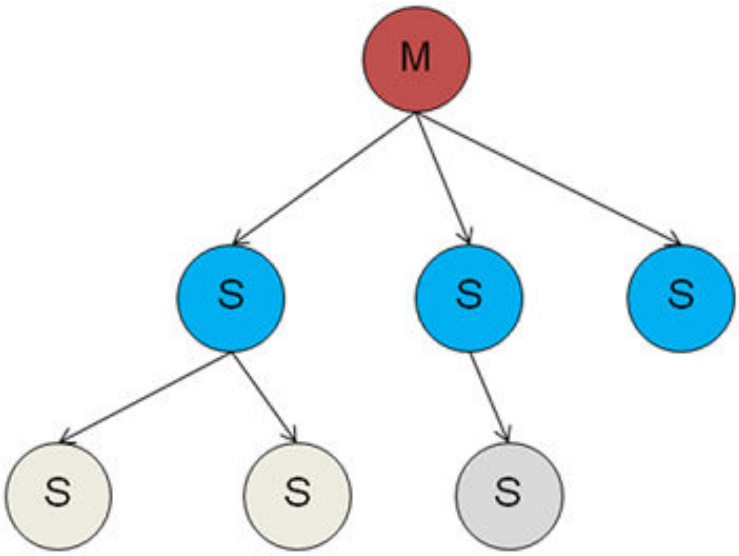
Redis实例可以部署在一个或多个节点上，且一个节点上也可以部署一个或多个Redis实例（FusionInsight HD平台中，每个节点上Redis实例的个数由软件根据节点硬件资源情况计算得出）。

最新版本的Redis支持集群功能，可以将多个Redis实例组合为一个Redis集群，从而对外提供一个分布式key-value数据库。集群通过分片（sharding）来进行数据共享，并提供复制和故障转移功能。

- 单实例模式

  单实例模式逻辑部署方式如图2-77所示：

图 2-77 单实例模式



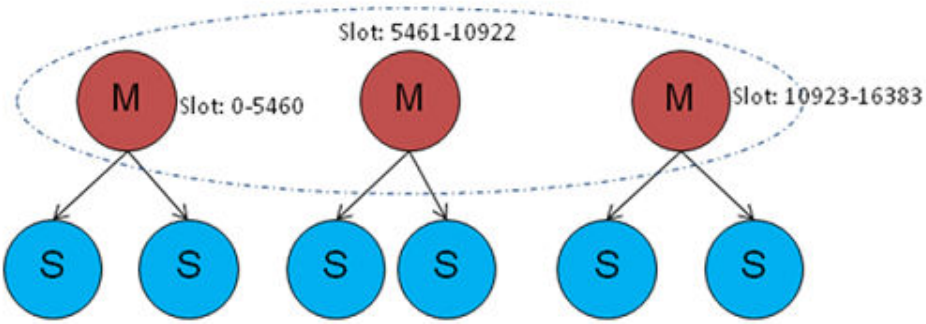
说明：

- 一个主实例（master）可以对应有多个从实例（slave），从实例本身还可连接从实例。
- 发给主实例的命令请求，主实例会实时同步给从实例进行处理。
- 主实例宕机，从实例不会自动升主。
- 从实例默认只读，在配置了“slave-read-only”为no时，从实例也可写。但从实例重启后，会从主实例同步数据，之前写入从实例的数据丢失。
- 多层级从实例的结构，相对所有从实例都直接连接在主实例下的结构，由于减少了主实例需要直接同步的从实例个数，一定程度上能提升主实例的业务处理性能。

- 集群模式

  集群模式逻辑部署方式如图2-78所示：

图 2-78 集群模式



说明：

- 多个Redis实例组合为一个Redis集群，共16384个槽位均分到各主实例上。

- 集群中的每个实例都记录有槽位与实例的映射关系，客户端也记录了槽位与实例的映射。客户端根据key算hash并模16384得到槽位，根据槽位-实例映射，将消息直接发送到对应实例处理。
- 默认情况，从实例不能读不能写，在线执行readonly命令可使从实例可读。
- 主实例故障，由集群中剩余的主实例选举出一个从实例升主，需要半数以上主实例OK才能选举。
- cluster-require-full-coverage配置项指示集群是否要求完整，若配置为yes，则其中一组主从都故障时，集群状态为FAIL，整个集群不能处理命令；若配置为no，则半数以上主实例OK，集群状态还是OK。
- Redis集群可以进行扩容、减容（新实例加入集群或Redis实例退出集群），并进行槽位迁移。
- 目前FusionInsight HD中的Redis集群只支持一主一从模式。

3 部署方案

关于本章

3.1 组网方案

基本概念

FusionInsight HD集群的组网方案中包含3种节点，如表3-1所示。

表 3-1 基本概念

概念	说明
管理节点	Management Node (MN)，用于安装FusionInsight Manager，即FusionInsight HD集群的管理系统。FusionInsight Manager对部署在集群中的节点及服务进行集中管理。
控制节点	Control Node (CN)，控制节点控制监控数据节点执行存储数据、接收数据、发送进程状态及完成控制节点的公共功能。 FusionInsight HD的控制节点包括HMaster、HiveServer、ResourceManager、NameNode、JournalNode、SlapdServer等。
数据节点	Data Node (DN)，执行管理节点发出的指示，上报任务状态、存储数据，以及执行数据节点的公共功能。 FusionInsight HD的数据节点包括DataNode、RegionServer、NodeManager、LoaderServer等。

典型组网

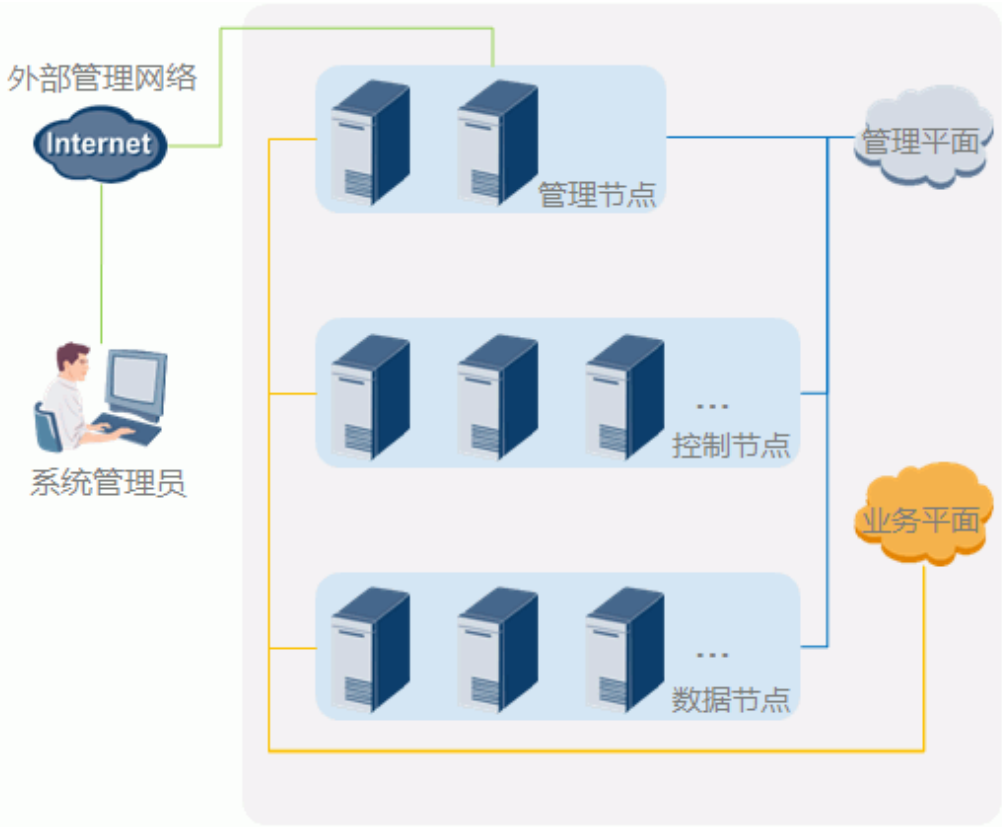
FusionInsight HD整个系统网络划分为2个平面，即业务平面和管理平面，两个平面之间采用物理隔离的方式进行部署，保证业务、管理各自网络的安全性。

- 业务平面通过业务网络接入，主要为用户和上层用户提供业务通道，对外提供数据存取、任务提交及计算的能力。
- 管理平面通过运维网络接入，提供系统管理和维护功能，主要用于集群的管理，对外提供集群监控、配置、审计、用户管理等服务。

主备管理节点还支持设置外部管理网络的IP地址，用户可以通过外部管理网络进行集群管理。

在典型配置下， FusionInsight HD集群采用双平面组网，如图3-1所示。

图 3-1 典型组网



组网原则

根据集群内节点数的规模，FusionInsight HD的组网原则如表3-2所示。

表 3-2 组网原则

节点部署原则	组网规则	适用场景
管理节点、控制节点和数据节点分开部署	集群内节点划分到不同子网，各子网通过核心交换机三层互联，每个子网的节点数控制在200个以内，不同子网中节点数量请保持均衡。	● （推荐）节点数大于200节点的集群使用此场景 ● 此方案至少需要8个节点
管理节点和控制节点合并部署，数据节点单独部署	集群内节点部署在同一子网，集群内通过汇聚交换机二层互联。	（推荐）节点数大于等于6，小于等于200节点的集群使用此场景
管理节点、控制节点和数据节点合并部署	集群内节点部署在同一子网，集群内通过汇聚交换机二层互联。	● 节点数小于6的集群使用此场景 ● 此方案至少需要3个节点 **说明** 不推荐使用此场景 ● 如节点数量满足需求，建议将数据节点单独部署。 ● 如节点数量不满足将数据节点单独部署的要求，必须使用此场景时，需要使用双平面组网方式。

3.2 硬件及运行环境要求

服务器配置要求

华为FusionInsight HD支持通用的X86服务器，企业可根据自身需求灵活选择。

安装FusionInsight HD的服务器配置要求如表3-3所示。

表 3-3 硬件最低要求

硬件	配置
CPU	● 最低配置：双路4核Intel处理器 ● 推荐配置：双路8核Intel处理器
Bit-mode	64位
内存	物理内存不少于64GB。为了满足实际业务运行，内存大小要求请结合实际服务部署计算规划，请参见部署原则。
网卡	● 管理平面使用两个GE电口配置bond。 ● 业务平面使用两个10GE光口配置bond。 ● 外部管理网络使用两个GE电口配置bond。

硬件	配置
磁盘RAID配置	管理节点和控制节点的非操作系统盘用来存放FusionInsight HD元数据，数据节点的非操作系统盘，用来存放FusionInsight HD的Hadoop数据。 ● 管理节点：操作系统盘和非操作系统盘分别独占一个RAID组，且RAID组级别为RAID1 ● 控制节点：操作系统盘和每个非操作系统盘分别独占一个RAID组，且RAID组级别为RAID1 ● 数据节点：操作系统所在盘独占一个RAID组，且RAID组级别为RAID1，非操作系统所在盘配置RAID0或无RAID **说明** 如果使用旧磁盘安装，请在安装前重新配置磁盘RAID并格式化磁盘。
磁盘数量	磁盘数量指配置RAID后操作性系统可见的磁盘数量。例如某节点有6块物理磁盘配置3组RAID1，则该节点磁盘数量为3。 如未安装全部服务，可减少对应服务的元数据盘或数据盘数，各服务需要的元数据盘或数据盘数量请参考《准备操作系统》。 ● 所有节点分设： – 管理节点：每个节点2个磁盘盘符（1个OS盘符、1个元数据盘符）。 – 控制节点：每个节点6个磁盘盘符（1个OS盘符、5个元数据盘符）。 – 数据节点：每个节点10个磁盘盘符（1个OS盘符、9个数据盘符）。 ● 管理节点和控制节点合设、数据节点分设： – 管理节点+控制节点：每个节点7个磁盘盘符（1个OS盘符、6个元数据盘符）。 – 数据节点：每个节点10个磁盘盘符（1个OS盘符、9个数据盘符）。 ● 所有节点合设：每个节点16个磁盘盘符（1个OS盘符、6个元数据盘符、9个数据盘符）。 **说明** ● 安装SolrServerAdmin的2个数据节点，每个节点需要增加1个数据盘盘符。 ● 如果Solr索引数据存放在HDFS上，每个数据节点可以减少5个数据盘盘符。 ● 每个Redis实例对应一个磁盘分区（Redis的实例个数为节点CPU核数-2），为保证集群性能，建议每个磁盘上分配的Redis分区数不大于5个。
磁盘空间	如何设置RAID容量请参见《软件安装》手册中“设置RAID1容量”章节。当磁盘容量为600G时，用户在磁盘故障后重建时间可控制在4小时之内。根据用户实际需求，可以灵活设置磁盘容量。 ● 管理节点：操作系统盘≥600GB，每个非操作系统盘≥600GB ● 控制节点：操作系统盘≥600GB，每个非操作系统盘≥600GB ● 数据节点：操作系统盘≥600GB，每个非操作系统盘≥500GB

运行环境要求

保证FusionInsight HD系统正常运行的本地环境要求如表3-4所示。

表 3-4 软件环境要求

软件	要求
操作系统	推荐： ● SUSE Linux Enterprise Server 11 SP3（SUSE11.3） ● RedHat-6.4-x86_64（Red Hat6.4） 可用： ● SUSE Linux Enterprise Server 11 SP1（SUSE11.1）、SUSE Linux Enterprise Server 11 SP2（SUSE11.2） ● RedHat-6.5-x86_64（Red Hat6.5）和RedHat-6.6-x86_64（Red Hat6.6） ● CentOS－6.6版本（CentOS6.6）、CentOS-6.5版本（CentOS6.5）和CentOS-6.4-x86_64（CentOS6.4）
浏览器	● Internet Explorer 9（标准模式）及以上版本 **说明** 如需使用Solr，请使用Internet Explorer 10以上版本浏览器。 ● Google Chrome 21及以上版本 ● Mozilla Firefox浏览器版本： － Windows：24.x/31.x/40.x及以上版本 **说明** Firefox 24.x版本需要执行以下动作可正常访问FusionInsight Manager: 1. 在地址栏输入“about:config”，按回车打开设置界面。 2. 搜索“security.tls.version.max”，双击将数值改成“3”。 － Linux： － Red Hat6.6自带的Firefox 31.1.0版本 － SUSE11.3自带Firefox 17.0.4版本 **说明** ● Hue不支持该版本浏览器。 ● 使用Linux自带Firefox浏览器的配置方法请参考《软件安装》的“使用Linux自带Firefox浏览器访问FusionInsight Manager”章节。
JDK	● Oracle JDK：服务端支持1.8版本，客户端支持1.7和1.8版本 ● IBM JDK：服务端支持1.8版本，客户端支持1.7和1.8版本

3.3 软件部署方案

软件清单

FusionInsight HD V100R002C60SPC200版本用到的开源组件版本号如表3-5所示。

表 3-5 软件清单

组件名称	版本号
Hadoop	2.7.2
HBase	1.0.2
Hive	1.3.0
ZooKeeper	3.5.1
Phoenix	4.4.0
Oozie	4.2.0
Hue	3.9.0
Spark	1.5.1
Sqoop	1.99.3
Flume	1.6.0
Kafka	2.10-0.9.0.0
Solr	5.3.1
Kerberos	1.10.7
Streaming	0.10.0
Redis	3.0.5

部署原则

FusionInsight HD各服务的部署原则如表3-6所示。

说明

- 集群中各服务之间存在依赖或者关联的关系：
  - A依赖于B表示，若集群中部署A服务，需要提前或同时部署B服务。A与B可以不部署在相同的节点上。
  - A与B关联表示，若集群中部署A服务，需要同时部署B服务。A与B需要部署在相同的节点上。
- 安装集群时只能安装一对NameNode和Zkfc，当HDFS服务设置HDFS Federation需要部署多对Namenode和Zkfc时，需要在集群安装完成后手动添加其余部分。

表 3-6 各服务角色的内存要求和部署原则

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
OMSServer	OMSServer	10GB	-	分别部署在2个管理节点上，主备配置。
LdapServer	SlapdServer	500MB-1GB	-	考虑性能最优化，建议所有集群中LS都与KrbServer部署在相同的节点上。 ● 分析集群：LS分别部署在2个控制节点上，主备配置。 ● 备份集群：LS分别部署在2个控制节点上，2个均为分析集群的备用服务。
KrbServer	KerberosServer	3MB	● KrbServer依赖于LdapServer ● KerberosServer与KerberosAdmin关联	分别部署在2个控制节点上，负荷分担。
	KerberosAdmin	2MB		分别部署在2个控制节点上，负荷分担。
ZooKeeper	QP（quorumpeer）	1GB	-	每个集群内配置3个在控制节点上。如需扩展，请保持数量为奇数个。
HDFS	NN（NameNode）	4GB	● NameNode与Zkfc关联 ● 依赖于ZooKeeper	分别部署在2个控制节点上，主备配置。
	Zkfc（ZooKeeper FailoverController）	1GB		分别部署在2个控制节点上，主备配置。
	JN(JournalNode)	4GB		至少部署3个在控制节点上，每个节点保留一份备份数据。如需保留超过三份以上备份，可部署多个在控制或数据节点上，请保持数量为奇数个。
	DN（DataNode）	4GB		至少部署3个，建议部署在数据节点上。
Yarn	RM（ResourceManager）	2GB	依赖于HDFS和ZooKeeper	分别部署在2个控制节点上，主备配置。
	NM（NodeManager）	2GB		部署在数据节点上，与HDFS的DataNode保持一致。

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
Mapreduce	JHS（JobHistoryServer）	2GB	依赖于Yarn、HDFS和ZooKeeper	1个集群内只能部署1个在控制节点上。
DBService	DBServer	512MB	-	分别部署在2个控制节点上，主备配置。
Hue	Hue	1GB	依赖于DBservice	分别部署在2个控制节点上，主备配置。
Loader	LS(LoaderServer)	2GB	依赖于MapReduce、Yarn、DBService、HDFS和ZooKeeper	● 分别部署在2个节点上，主备配置。 ● Loader必须部署在任意的2个NodeManager节点上。
Spark	SR（SparkResource）	-	依赖于Yarn、Hive、HDFS、Mapreduce、ZooKeeper和DBService	无实体进程，不消耗内存。所有数据节点上都要部署，非主备。
	JH（JobHistory）	2GB		分别部署在2个控制节点上，非主备。
	JS（JDBCServer）	2GB		分别部署在2个控制节点上，主备配置。
Hive	HS（HiveServer）	4GB	依赖于DBService、Mapreduce、HDFS、Yarn和ZooKeeper	至少部署2个在控制节点上。可部署多个在控制节点上，负荷分担。
	MS（MetaStore）	2GB		至少部署2个在控制节点上。可部署多个在控制节点上，负荷分担。
	WebHCat	2GB		至少部署1个在控制节点上。可部署多个在控制节点上，负荷分担。
HBase	HM（HMaster）	1GB	依赖于HDFS、ZooKeeper和Yarn	分别部署在2个控制节点上，主备配置。
	RS（RegionServer）	6GB		部署在数据节点上，与HDFS的DataNode保持一致。
	TS（ThriftServer）	1GB		每个集群部署3个在控制节点上。若ThriftServer访问HBase延时不能满足用户需求的时候，可以部署多个在控制或者数据节点上。

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
FTP-Server	FTP-Server	1GB	依赖于HDFS和ZooKeeper	每个实例默认提供16个并发通道，当所需并发数更大时，可部署最多8个在控制节点或数据节点上。
Flume	Flume	1GB	依赖于HDFS和ZooKeeper	建议单独部署，不建议Flume和DataNode部署在同一节点，否则会存在数据不均衡的风险。
	MonitorServer	128MB		分别部署在2个控制节点上，非主备。
Kafka	Broker	1GB	依赖于ZooKeeper	至少部署2个在数据节点上。若每天产生的数据量超过2TB，建议部署多个在数据节点上。
Metadata	Metadata	512MB	依赖于DBService	1个集群内只能部署1个在控制节点上。
Oozie	oozie	1GB	依赖于DBService、Yarn、HDFS、MapReduce和ZooKeeper	分别部署在2个控制节点上，主备配置。
Solr	SolrServerN (N=1～5)	2GB	依赖于ZooKeeper	每个节点可以配置5个实例。建议配置3个以上节点，每个节点上的实例个数均匀分布。 ● 数据条目在10亿以下时，建议配置3个实例在数据节点上。 ● 数据条目在10亿～20亿时，建议配置8～12个实例在数据节点上。 ● 数据条目在20亿以上时，建议配置在3个或以上独立与DataNode所在节点之外的数据节点上，每个节点配置5个实例。

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
	SolrServerAdmin	2GB	**说明** ● Solr数据选择保存在HDFS上时，还依赖于HDFS。 ● Solr数据优先选择保存到HDFS，每个节点部署三个Solr实例。 ● 单节点实时索引速度2MB/s以上的，建议部署到本地磁盘，每个节点Solr实例部署5个，每个Solr实例单独挂载磁盘。 ● 存储HDFS相比本地磁盘性能下降30%~50%。	分别部署在2个数据节点上，非主备。
	HBaseIndexer	512MB	依赖于HBase、HDFS和ZooKeeper	建议每个SolrServer实例所在节点均部署一个。
SmallFS	FGCServer	6GB	依赖于MapReduce、YARN、HDFS和ZooKeeper	分别部署在2个控制节点上，主备配置。
Streaming	Logviewer	256MB	-	根据Supervisor的部署情况进行规划，每个部署Supervisor的节点也需要部署Logviewer。
	Nimbus	1GB	依赖ZooKeeper	分别部署在两台控制节点上，主备配置，和UI是联动关系。
	UI	1GB	依赖ZooKeeper	分别部署在两台控制节点上，和Nimbus是联动关系（部署Nimbus节点肯定会有UI）。

服务名称	角色名称	内存最小要求	依赖关系	角色业务部署原则
	Supervisor	1GB	-	至少部署在一台控制节点或数据节点上，如需提供大量计算能力，可部署多个，建议单独部署在控制节点上。主要负责管理工作进程（Worker），工作进程数量、内存可配置，对资源占用较大。 **说明** Supervisor部署数目可根据如下公式进行计算，其中拓扑数和每个拓扑要求的Worker数为客户自行规划项，Supervisor配置的Worker数默认为4。 需要的Supervisor数=拓扑数 × 每个拓扑要求的Worker数 / Supervisor 配置的Worker数
Redis	Redis_1、Redis_2、Redis_3……	1GB	依赖DBService	单master模式Redis至少部署在一台数据节点上，如需部署Redis集群，则至少要部署到三台数据节点上。

4 企业级增强特性

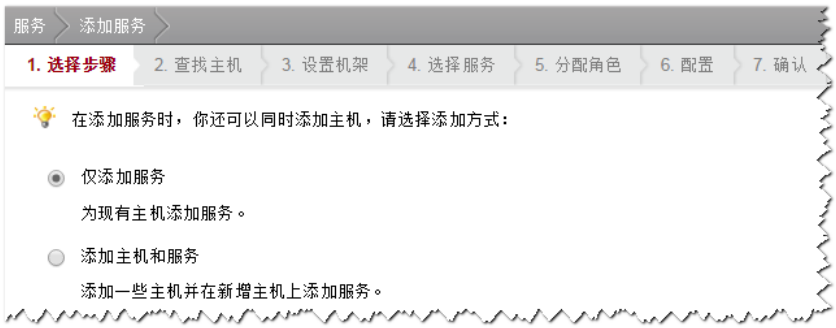
关于本章

4.1 Manager

一键式安装

Manager的WebUI提供向导式的集群安装步骤，根据界面用户可一步步完成集群安装。

图 4-1 集群一键式安装



统一监控告警

Manager提供可视化、便捷的监控告警功能。通过“系统概览”，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实时上报告警，界面帮助提供性能指标和告警恢复的详细方法，帮助用户快速解决故障。

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。

单点登录

提供Manager WebUI与组件WebUI之间的单点登录，以及FusionInsight HD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。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，页面将自动锁定。

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。

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FusionInsight HD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。

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。

4.2 KrbServer 及 LdapServer

集群内服务认证

在使用安全模式的FusionInsight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

FusionInsight各组件提供了应用开发接口，用于客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。

LdapServer HA 机制

FusionInsight中集成的LdapServer服务提供了HA机制，提升了LdapServer的高可用性。

跨集群互信特性

FusionInsight提供两个集群之间的互信功能，用于实现集群之间的数据读、写等操作。

4.3 HBase

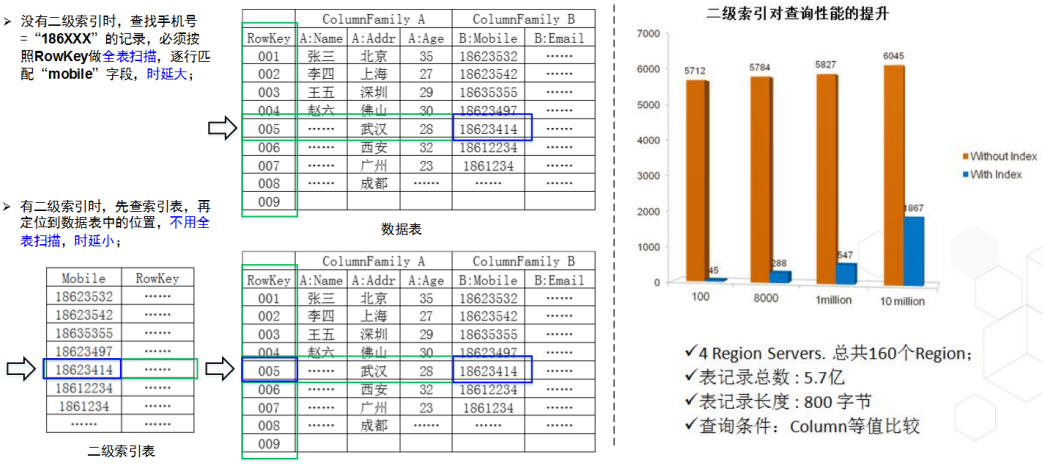
支持二级索引

HBase是一个Key-Value类型的分布式存储数据库。每张表的数据，是按照RowKey的字典顺序排序的，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二级索引产生的背景。二级索引为HBase提供了按照某些列的值进行索引的能力。

图 4-2 二级索引



二级索引 rowkey 去除 padding

二级索引rowkey是由starkey of index region + index name + indexed column(s) value(s) + user table rowkey构成。

在此版本的之前版本中，每一个字段是定长的， indexed column(s) value(s) 该字段由列值最长的决定。在一些已用场景中如果90%列值都很短，但是10%列值很长，这样会存

在很大的存储空间浪费。为了节省空间，采用分隔符分割每个字段而不是采用padding的方式。

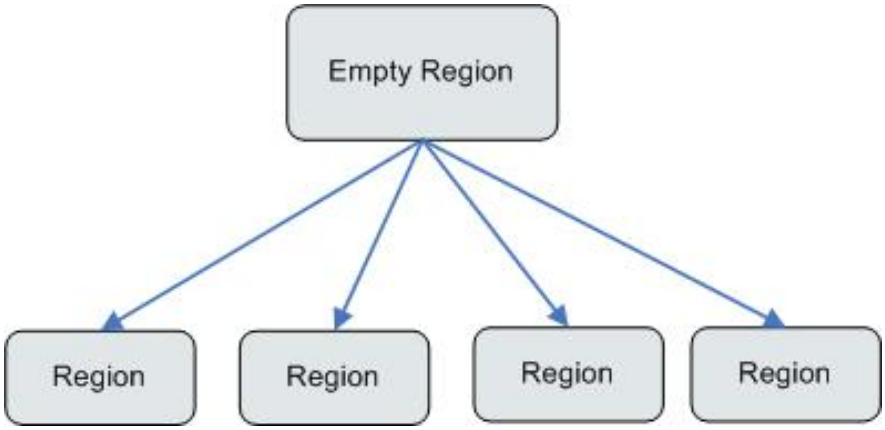
支持多点分割

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 4-3 多点分割



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。

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BukLoad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*使用该过滤其实现ACL同步。

4. 备集群只读限制，备集群只接受备集群节点内的超级用户对备集群的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。

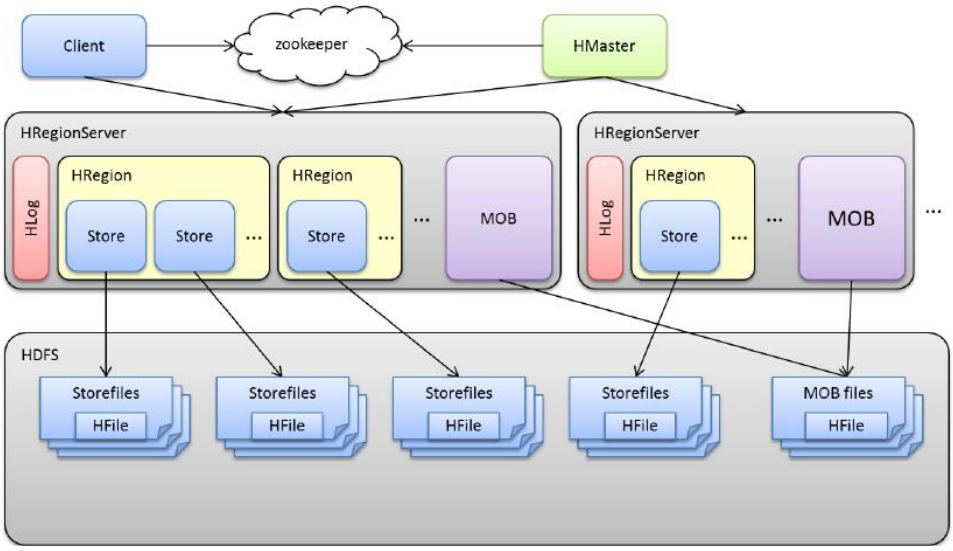
HBase MOB

在实际应用中，用户需要存储大大小小的数据，比如图像数据、文档。小于10MB的数据一般都可以存储在HBase上，对于小于100KB的数据，HBase的读写性能是最优的。如果存放在HBase的数据大于100KB甚至到10MB时，插入同样个数的数据文件，其数据量很大，会导致频繁的compaction和split，占用很多CPU，磁盘IO频率很高，性能严重下降。

将MOB数据（即100KB到10MB大小的数据）直接以HFile的格式存储在文件系统上（例如HDFS文件系统），然后把这个文件的地址信息及大小信息作为value存储在普通HBase的store上，通过expiredMobFileCleaner和Sweeper工具集中管理这些文件。这样就可以大大降低HBase的compation和split频率，提升性能。

如图4-4所示，图中MOB模块表示存储在HRegion上的mobstore，mobstore存储的是key-value，key即为HBase中对应的key，value对应的就是存储在文件系统上的引用地址以及数据偏移量。读取数据时，mobstore会用自己的scanner，先读取mobstore中的key-value数据对象，然后通过value中的地址及数据大小信息，从文件系统中读取真正的数据。

图 4-4 MOB 数据存储原理



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FusionInsight HD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。

4.4 HDFS

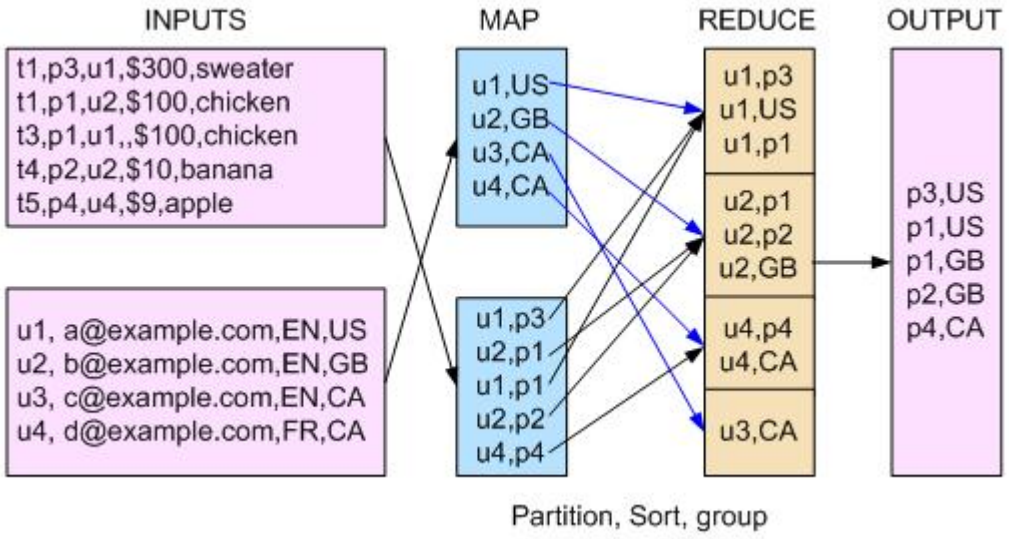
文件块同分布（Colocation）

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4-5所示：

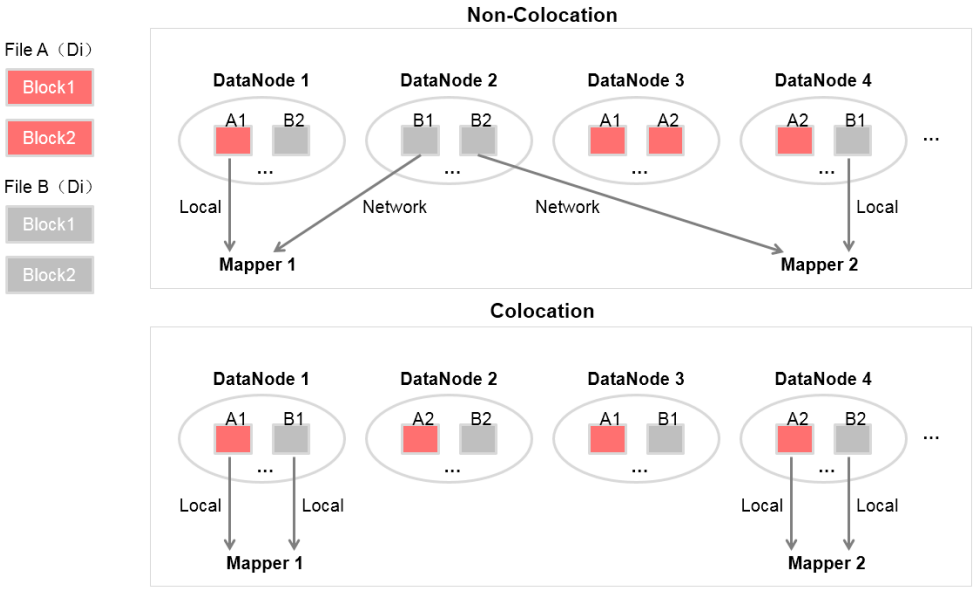
图 4-5 无同分布数据传输流程



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 4-6 无同分布与同分布数据块分布对比

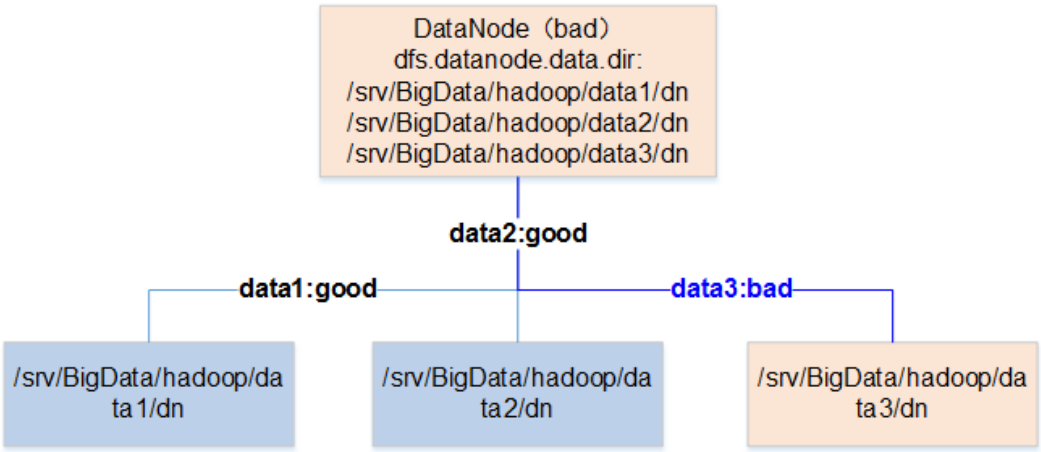


硬盘坏卷设置

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。

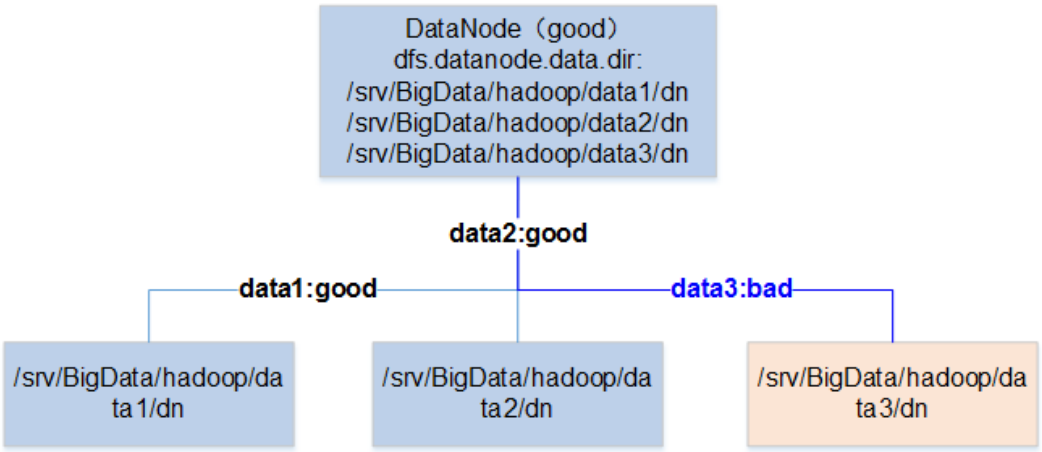
“dfs.datanode.failed.volumes.tolerated”取值范围为大于等于0，默认值为0，效果如图4-7所示。

图 4-7 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4-8所示。

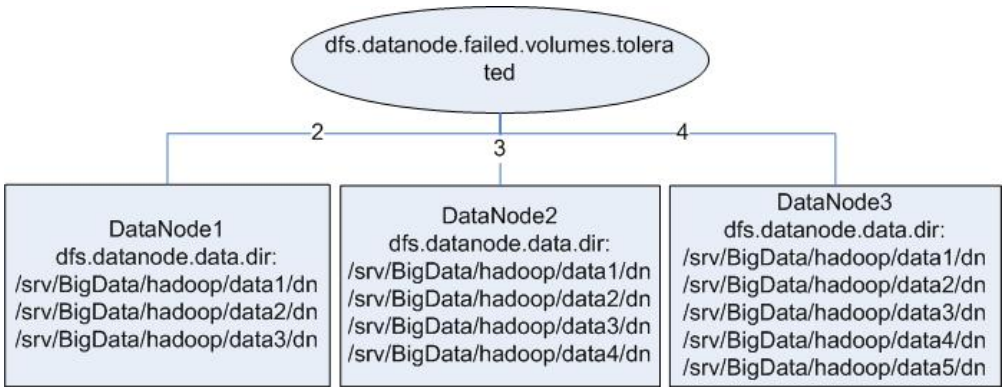
图 4-8 选项设置为 1



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5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4-9所示进行设置。

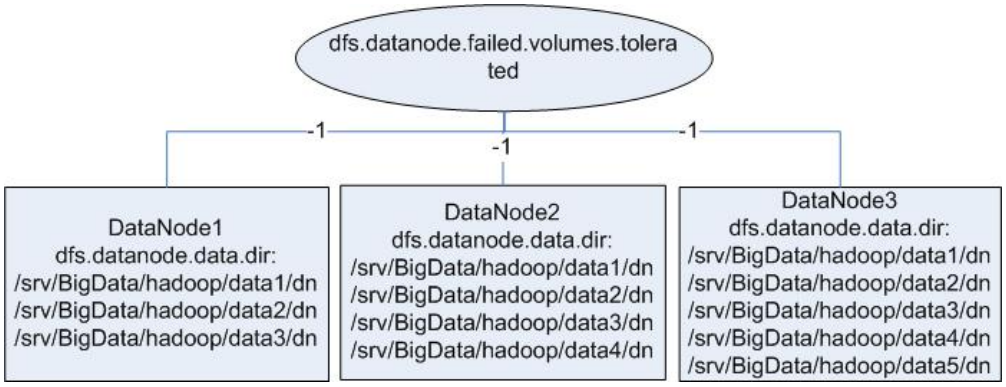
图 4-9 未增强前属性设置



在华为版本的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4-10所示。

图 4-10 增强后属性配置



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。

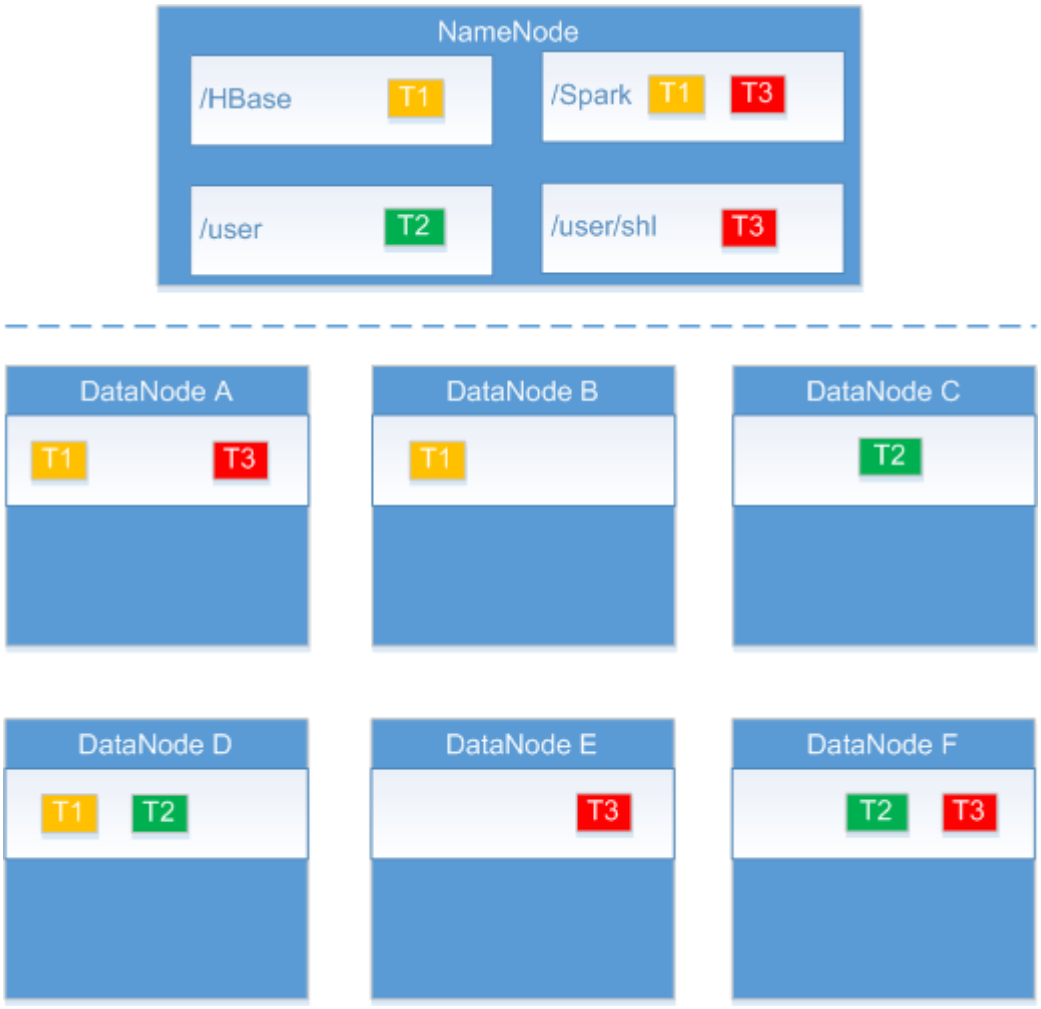
在开源HDFS中，fsimage里保存了所有类型的元数据信息，每一类元数据信息（如文件元数据信息和文件夹元数据信息）分别保存在一个section块里，这些section块在启动时是串行加载的。当HDFS上存储了大量的文件和文件夹时，这两个section的加载就会非常耗时，影响HDFS文件系统的启动时间。华为HDFS NameNode在生成fsimage时可以将同一类型的元数据信息分段保存在多个section里，当NameNode启动时并行加载fsimage中的section以加快加载速度。

基于标签的数据块摆放策略

用户需要通过数据特征灵活配置HDFS数据块摆放策略，即一个HDFS目录对应一个标签表达式，每个DataNode可以对应一个或多个标签；当基于标签的数据块摆放策略为指定目录下的文件选择DataNode节点进行存放时，根据文件的标签表达式选择出将要存放的DataNode节点范围，然后在这个DataNode节点范围内，遵守下一个指定的数据块摆放策略进行存放。如图4-11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 4-11 基于标签的数据块摆放策略样例



4.5 YARN

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。

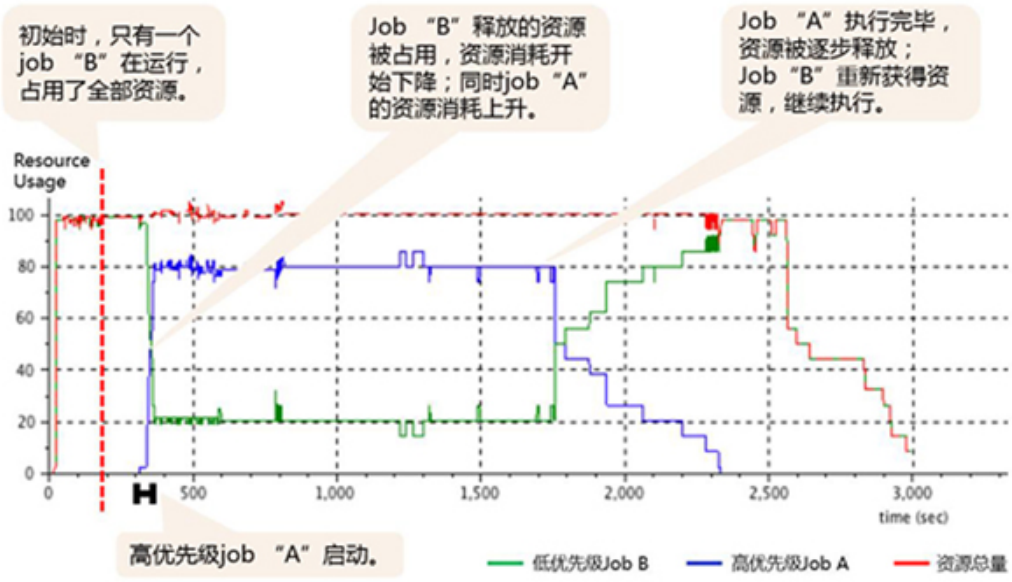
华为提供了任务优先级调度机制。此特性允许用户定义不同优先级的Job，后启动的高优先级Job能够获取运行中的低优先级Job释放的资源；低优先级Job未启动的计算容器被挂起，直到高优先级Job完成并释放资源后，才被继续启动。

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

容器可重用与任务优先级调度有冲突，如果启用了容器重用，资源就不会被释放，那么优先级调度就不起作用了。

图 4-12 任务优先级调度



提交 Application 可设置超时参数

开源功能：MapReduce任务在运行时，如果执行很长时间，会被一直挂起。用户只能等待，且不能判断其原因。

现增加一个超时参数***-Dapplication.timeout.interval*** =，用户在提交MapReduce任务时，可设置此超时参数，该参数的单位是秒。当任务运行时间超过指定的时间后，会停止此任务。

yarn jar<*App_Jar_Name*> *[Main_Class]* ***-Dapplication.timeout.interval*** = <*timeout*>

说明

该参数值需指定为整数。如果该参数未设置，则超时功能不会执行。如果该参数设置了一个无效值（即非整数），则使用默认值5分钟。

4.6 MapReduce

容器可重用

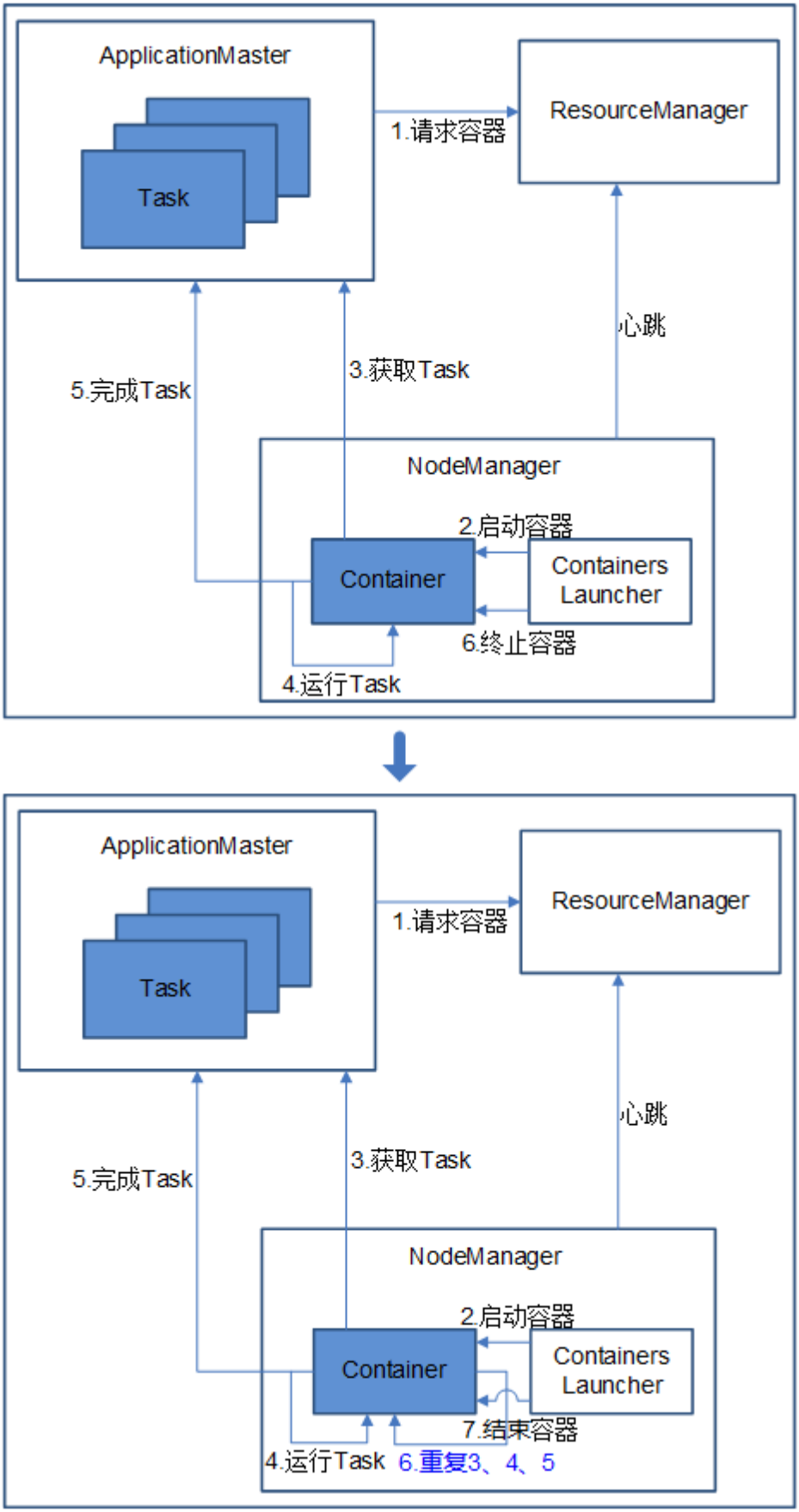
原生MapReduce将一个作业（Job）分解为若干个任务（Task）来执行，执行的载体在MapReduce内部称为容器（Container），它表示一个计算能力单元，物理上是一个动态运行的JVM进程。

在一个Task完成后，会终止Container，ApplicationMaster会通过资源请求重新分配容器，进行初始化，运行新的Task。而容器重用特性允许容器在一个Task运行完成后自动去获取新的Task，避免了容器重新分配以及初始化动作，大大减少了容器启动与回收时间，从而提升Job执行效率。优化后的MapReduce集群计算性能得到有效提升。

容器重用只能在同类型任务之间重用，例如：多个map任务之间、多个reduce任务之间重用；不能跨越不同类型任务进行重用，例如：map和reduce之间不能重用。

社区的Hadoop1.0版本实现了该特性，但Hadoop2.0版本没有实现，并且华为的实现方式和社区Hadoop1.0版本也不同，华为提供的方案可以通过设置参数更好的满足MapReduce本地化的需求。

图 4-13 容器可重用



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 4-14 MapReduce job

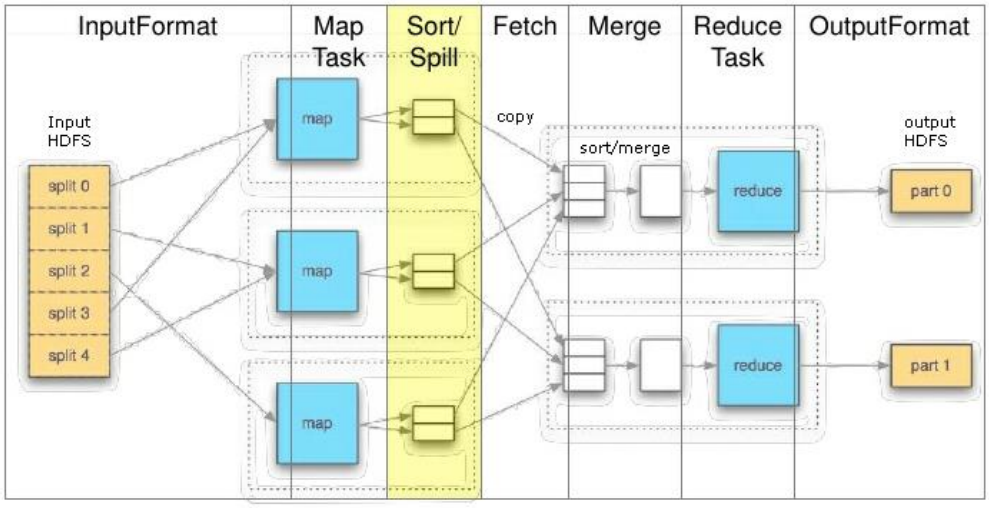
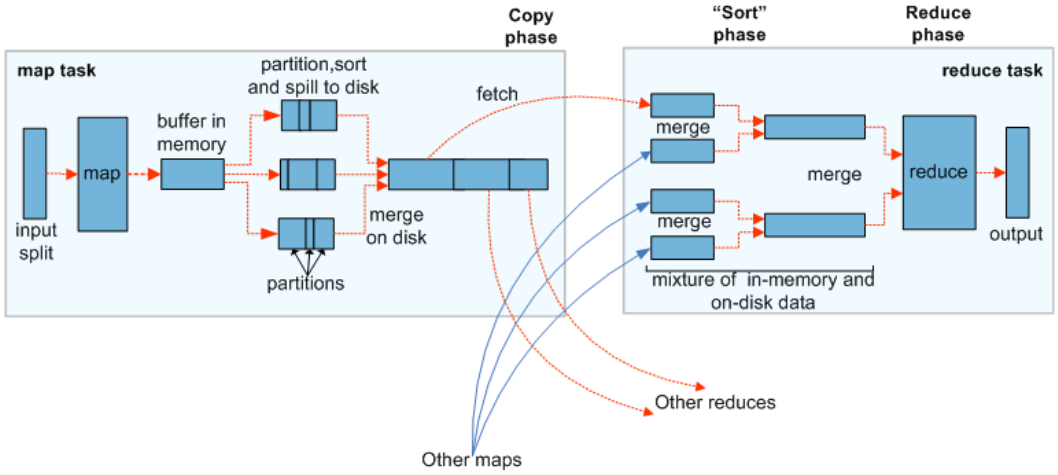


图 4-15 MapReduce job execution flow



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。

MR 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MR 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MR 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MR 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

1. 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。
2. 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。

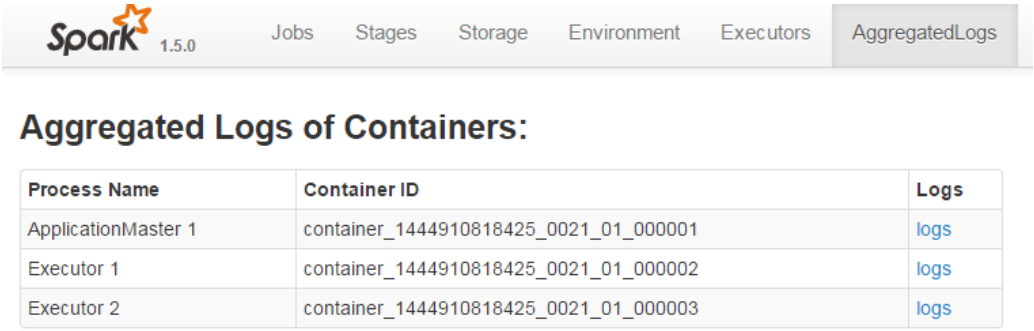
4.7 Spark

4.7.1 Spark Core 增强

WebUI 增加 Spark 应用日志

当YARN配置“yarn.log-aggregation-enable”为“true”时，就开启了container日志聚合功能。开启container日志聚合功能之后，其日志聚合至HDFS目录中，只能通过获取HDFS文件来查看日志。WebUI增加Spark应用日志功能则是针对聚合日志实现了在Web界面上直接查看日志的功能。该功能只有在开启聚合日志功能后才能在WebUI上看见相应的标签页，其界面如图4-16所示：

图 4-16 聚合日志显示页面



在界面上针对特定的Job依次显示ApplicationMaster和各Executor的日志，其排序规则先container名字（ApplicationMaster或Executor）后序号。点击对应的logs链接便会在界面上直接展示相应的聚合日志。

WebUI增加Spark应用日志通过MapReduce的JobHistoryServer来解析聚合日志的，因此需保证已完成MapReduce服务，且JobHistory角色运行正常。

DAG 打印

DAG（无回路有向图）能够清晰的描述一个Spark Job的执行流程，通过DAG可以较快的分析出Job的执行流程是否合理，从而快速进行优化。

由于在Spark中只有RDD执行action时才会触发RDD的计算，而每个RDD的action都会调用SparkContext中的runJob函数，因此在该函数中添加DAG打印的逻辑。设置SparkConf的属性spark.logLineage为true，可以将DAG打印到日志中。

DAG打印格式如下：

1. 每一行打印一个RDD，包含类型、ID、在代码中生成位置等信息；
2. 每个RDD的下一行为其父RDD，若父RDD有多个，则缩进2字符，并在该行行首打印标识字符串。

图 4-17 DAG 打印示例

```
15/01/26 22:38:37 INFO SparkContext: RDD.toDebugString:
 (12) MappedRDD[13] at map at dagPrint.scala:26 []
 |   MappedRDD[12] at map at dagPrint.scala:25 []
 |   FlatMappedValuesRDD[11] at join at dagPrint.scala:24 []
 |   MappedValuesRDD[10] at join at dagPrint.scala:24 []
 |   CoGroupedRDD[9] at join at dagPrint.scala:24 []
 |   ShuffledRDD[4] at reduceByKey at dagPrint.scala:17 []
 +-(2) MappedRDD[3] at map at dagPrint.scala:16 []
    |  MappedRDD[2] at map at dagPrint.scala:15 []
    |  /data.txt MappedRDD[1] at textFile at dagPrint.scala:14 []
    |  /data.txt HadoopRDD[0] at textFile at dagPrint.scala:14 []
 +-(2) MappedRDD[8] at map at dagPrint.scala:22 []
    |  MappedRDD[7] at map at dagPrint.scala:21 []
    |  /data2.txt MappedRDD[6] at textFile at dagPrint.scala:20 []
    |  /data2.txt HadoopRDD[5] at textFile at dagPrint.scala:20 []
```

4.7.2 Spark SQL 增强

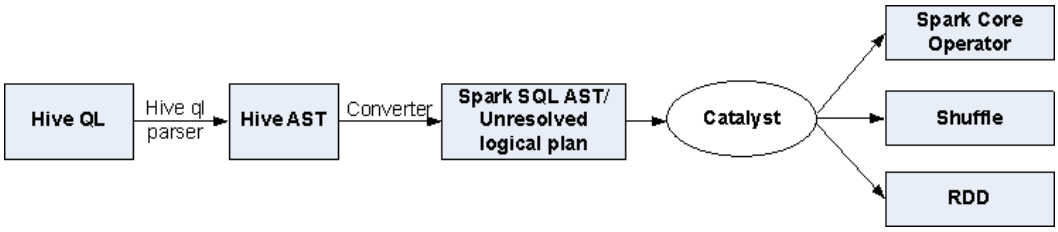
Spark SQL 能力

Spark SQL能力增强主要通过以下两个方面来体现：Hive语法兼容性、标准SQL语法兼容。

Hive语法兼容性

SparkSQL提供了一个HiveContext，支持部分Hive语法，其基本架构为：

图 4-18 HiveContext 基本架构



- Hivecontext使用Hive原生parser，SparkSQL自实现转换器将Hive parser解析出来的语法树（AST）转换为catalyst中的原生logical plan。
- Hivecontext继承自SQLcontext，其逻辑计划本质上走的都是catalyst，只有SQL parser使用了Hive的parser。

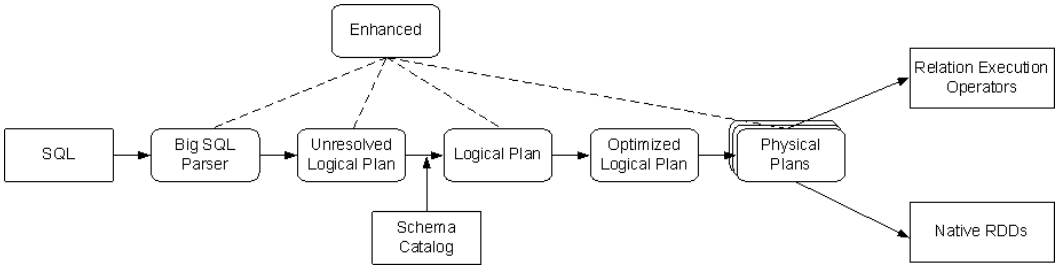
Hive语法兼容性以通过Hive-Test-benchmark测试集上的64个SQL语句为准，与Apache Spark相比，增加如下字段的兼容性：

表 4-1 Hive 语法兼容性新增支持字段

字段	描述
支持in中嵌套子查询	IN操作符支持用户选取某个字段属于某个集合的记录，当前不支持该集合通过子查询获得。
支持over	创建一个窗口，须要与排序函数或聚合函数一起使用。
支持with	with语句可以定义一个SQL片断，相当于一个临时变量，将一个SQL语句别命名，增强SQL语句的可读性。
支持exists子查询	exists中可以接一个子查询，作为where中的判断条件。

标准SQL语法兼容性

图 4-19 标准 SQL 语句执行流程



标准SQL增强遵循不对现有catalog和SQLcontext核心模块代码做修改的原则，具体实施时，在**Hivecontext**里面实现插件式的增强，具体如下：

- SQL parser：实现插件式的SQL parser实现，然后基于插件式parser接口实现99语法对应的parser。
- Logical Plan：新增一个SQL99LogicalPlans的类，所有新增的逻辑计划都由该类管理，不在原有逻辑计划中修改和添加。
- Expressions：新增一个SQL99Expressions，所有新增的expression都添加到该文件统一管理，不在原有expression中修改和添加。
- Optimizer：原则上优化器增强不在语法增强方案中体现，但是如若涉及优化规则添加，新增SQL99OptimizerRules。
- Physical plan：新增SQL99SparkPlans，所有新增的执行计划都添加到该文件统一管理，不在原有执行计划中修改和添加。
- 元数据管理：使用Hive metastore进行元数据管理。

标准SQL语法的兼容性以通过标准语法的tpc-ds测试集上的99个SQL语句为准，与Apache Spark相比，增加如下字段的兼容性：

表 4-2 标准 SQL 语法兼容性新增支持字段

字段	描述
支持top关键字	top类似于limit，用于返回查询结果的前n条记录。
支持with关键字	with语句可以定义一个SQL片断，相当于一个临时变量，将一个SQL语句别命名，增强SQL语句的可读性。
支持rank over	根据over指定的分区信息排序。
支持as后面字符串常量	将某个字段别命名为一个字符串里的内容。
cast支持日期转换	将其他类型的数据转换为日期数据类型。
order by排序关键字支持	当前SQLContext，要么对order by字句中的字段全加排序关键字（desc或asc），要么都使用默认的，而不能部分加关键字，没加关键字的就使用默认的排序关键字。
支持rollup	如果是Group by ROLLUP(A,B,C)的话，首先会对(A、B、C)进行GROUP BY，然后对(A、B)进行GROUP BY，然后是(A)进行GROUP BY，最后对全表进行GROUP BY操作，即多级汇总。
支持grouping	指明在group by子句中的某个表达式是否被聚合了，聚合了返回1，否则返回0。
order by中支持聚合函数表达式	支持在order by中根据聚合函数计算出来的结果进行排序。
支持exists和not exists	exists中可以接一个子查询，作为where中的判断条件。
in中支持子查询	判断某个字段或表达式是否存在于某个子查询的结果中。
支持over	创建一个窗口，需要与排序函数或聚合函数一起使用。
支持coalesce	返回参数列表中的第一个非空值。

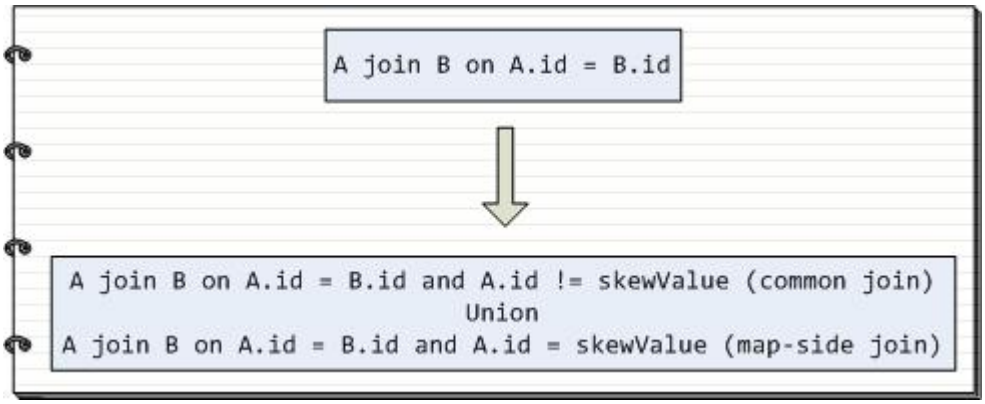
字段	描述
from字句后接复杂子查询	当前from语句后面接子查询支持不完善，from字句中连续接两个括号会出错。
支持stddev_samp	计算样本标准差。
where中支持子查询	支持某个字段与某一个子查询查询出来的结果比较。
支持dec	将二进制字符串转换为数字。
case when中支持子查询	支持某一个子查询查询出来的结果与其他值比较。
支持||操作符	字符串连接操作符。
支持numeric	将二进制字符串转换为数字。

数据倾斜优化

在Spark SQL多表Join的场景下，会存在关联键严重倾斜的情况，导致Hash分桶后，部分桶中的数据远远高于其他分桶。最终导致部分Task过重，运行得很慢；其他Task过轻，运行得很快。一方面，数据量大Task运行慢，使得计算性能低；另一方面，数据量少的Task在运行完成后，导致很多CPU空闲，造成CPU资源浪费。

针对数据倾斜的场景将原始的Join拆分成用于处理不倾斜关联键的Common Join和用于处理倾斜关联键的Map-Side Join(Broadcast Join)，如图4-20所示：

图 4-20 倾斜数据 Join 转换



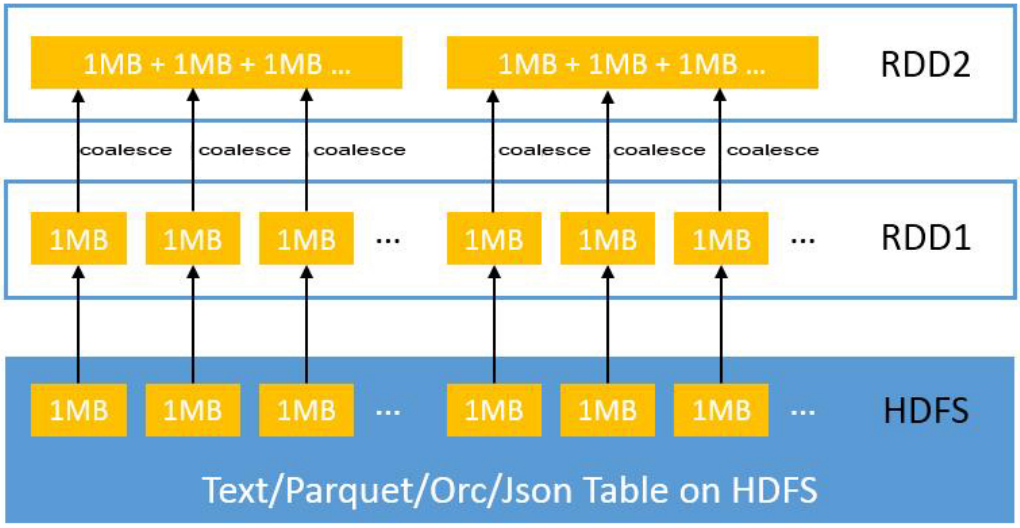
通过这样转换以后对于不包含倾斜键的数据依然能够平均到不同的Task进行处理，对于包含倾斜键的数据，将数据较小的那部分进行广播，利用Map-Side Join来平均到不同的Task进行处理，从而充分利用CPU资源，提升整体的性能。

小文件优化

Spark SQL的表中，经常会存在很多小文件（大小远小于HDFS块大小），每个小文件默认对应Spark中的一个Partition，也就是一个Task。在很多小文件场景下，Spark会起很多Task。当SQL逻辑中存在Shuffle操作时，会大大增加hash分桶数，严重影响性能。

针对小文件的场景采用coalesce算子，对Table中的小文件生成的partition进行合并，减少partition数，从而避免在shuffle的时候，生成过多的hash分桶，提高性能， 如图4-21所示：

图 4-21 小文件合并



权限管理

在安全集群中，SparkSQL提供的一套管理控制用户操作数据库的权限系统，以保证不同用户之间操作数据库的独立性和安全性。如果一个用户想操作另一个用户的表、数据库等，需要获取相应的权限才能进行操作，否则会被拒绝。

- 针对已有的Hive数据，SparkSQL的权限控制沿用Hive的机制和元数据，具备Hive数据访问权限的用户，可使用SparkSQL继续访问。
- SparkSQL的权限模型由元数据权限与HDFS文件权限组成。用户使用SparkSQL服务进行sql操作，必须对SparkSQL数据库和表（含外表和视图）拥有相应的权限。

4.8 ZooKeeper

日志增强

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。

4.9 Hive

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。

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)
- SMS4(对应加密类名称为：org.apache.hadoop.hive.serde2.SMS4Rewriter)

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，FusionInsight HD 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可以将自己的HBase表中符合条件的一条或者多条数据清除。

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割其规则文本。

FusionInsight HD 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。

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。

FusionInsight HD 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。

支持开启 Transform 功能

Hive开源社区版本禁止Transform功能。 FusionInsight HD Hive提供配置开关，默认为禁止Transform功能，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。

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。 FusionInsight HD 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。

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。FusionInsight HD 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。

支持列授权

Hive开源社区版本只支持表级别的权限控制。FusionInsight HD 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。

4.10 FTP-Server

支持 Kerberos 鉴权认证

Apache FTP Server的鉴权认证是将用户名和密码记录在文件中或者数据库。在分布式的系统中，这种存储方式存在一定的缺陷。对于文件存储的方式，不适合使用在分布式系统中，而对于数据库的存储方式，其和HDFS文件系统的用户管理的结合存在很大的不同。因此，FusionInsight HD采用集群中的Kerberos服务进行鉴权认证，使用户管理和集群用户管理和HDFS用户管理无缝集成。

FTP 传输文件到 HDFS 文件系统

以HDFS作为FTP-Server的存储文件系统，所有的数据都存放HDFS。

4.11 Loader

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：

1. 提供数据转化功能
2. 支持图形化配置转换步骤
3. 支持从SFTP/FTP服务器导入数据到HDFS
4. 支持从SFTP/FTP服务器导入数据到HBase表
5. 支持从SFTP/FTP服务器导入数据到Phoenix表
6. 支持从SFTP/FTP服务器导入数据到Hive表
7. 支持从HDFS导出数据到SFTP/FTP服务器
8. 支持从HBase表导出数据到SFTP/FTP服务器
9. 支持从Phoenix表导出数据到SFTP/FTP服务器
10. 支持从关系型数据库导入数据到HBase表
11. 支持从关系型数据库导入数据到Phoenix表
12. 支持从关系型数据库导入数据到Hive表
13. 支持从HBase表导出数据到关系型数据库
14. 支持从Phoenix表导出数据到关系型数据库
15. 支持从Oracle分区表导入数据到HDFS
16. 支持从Oracle分区表导入数据到HBase表
17. 支持从Oracle分区表导入数据到Phoenix表
18. 支持从Oracle分区表导入数据到Hive表

19. 支持从HDFS导出数据到Oracle分区表
20. 支持从HBase导出数据到Oracle分区表
21. 支持从Phoenix表导出数据到Oracle分区表
22. 在同一个集群内，支持从HDFS导数据到HBase、Phoenix表和Hive表
23. 在同一个集群内，支持从HBase和Phoenix表导数据到HDFS
24. 导入数据到HBase和Phoenix表时支持使用bulkload和put list两种方式
25. 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
26. 支持从HDFS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
27. 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
28. 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
29. 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）
30. 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
31. 支持批量导入/导出ETL任务
32. 支持ETL任务分页查询、关键字查询和分组管理
33. 对外部组件提供浮动IP

4.12 Flume

提升传输速度

可以配置将指定的行数数据作为一个Event，而不仅仅是一行，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Outofmemory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。

4.13 Metadata

元数据标签

MDM可以为所有抽取出来的元数据对象打标签，从而为后续的搜索功能以及数据血缘分析等扩展功能提供依据。

元数据差异比较

MDM可以为每次抽取上来的元数据提供差异变化信息记录，从而能对下层数据源的变迁过程进行记录，为后续可能的数据演进分析提供依据。

备份与恢复

MDM的元数据存储在FusionInsight HD可靠的组件DBService上，组件本身运行并不存在过程数据，所以基于可靠的DBService的备份与恢复能力，使得MDM的数据内容的备份与恢复更可靠。

4.14 Solr

Solr的华为增强特性主要包括以下几个方面：

- 实现HA与浮动IP的机制，提高Solr服务的可靠性。
- 增加了kerberos认证，保障了索引数据的安全性。

4.15 Streaming

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Storm集群中Nimbus出现单点故障，从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。

4.16 Redis

完善的集群管理功能

FusionInsight HD提供完善的Redis集群管理功能，通过FusionInsight Manager，用户可以将Redis实例组建为Redis集群，提升系统处理能力，同时满足高可靠性要求。

- 向导式创建Redis集群系统

图 4-22 创建 Redis 集群

FusionInsight HD支持一主一从模式的Redis集群，系统自动计算节点上可安装的Redis实例个数并分配主从关系。

- 集群扩容、减容

  当集群需要提供大规模的处理能力时，可以一键式扩容一对或多对主从实例。在此过程中，系统会自动完成数据迁移和数据平衡，用户无需关注。

- Balance

  出现扩容异常、部分实例掉线等异常场景时，Redis集群中的数据可能会分布不均匀，此时可以通过管理界面上提供的Balance功能，让系统自动对集群数据进行平衡，保证集群的健康运行。

- 性能监控与告警

  系统提供Redis集群的性能监控功能，可以通过直观的曲线图方式，了解当前Redis集群、实例的TPS吞吐量情况。

  系统为Redis集群提供了多种告警，例如集群下线告警、持久化失败告警、槽位分布不均告警、主备倒换事件、集群高可靠性受损告警等，甚至主从实例内存大小不一致都可以自动上报告警。丰富的告警帮助用户更加轻松的进行Redis集群的监控和管理。

集群可靠性保证

Redis社区提供ruby脚本的集群管理工具redis-trib.rb，创建集群时主从分配只能按照固定顺序排列，且无法保证集群的高可靠性。如果主实例和从实例在同一台机器上，则这种主从备份的意义不大，或者集群内一个主机发生故障将导致整个集群不可用。

FusionInsight HD在创建Redis集群的时候，能够根据用户选择的实例范围，自动进行计算，按照主机级高可靠原则来部署集群，同时在进行扩容和减容的操作时，仍然会保证该原则。这样可以保证集群内任意一台主机发生故障，集群都能够通过主从实例倒换来保证集群继续工作。

同时如果发生了部分节点、实例故障，整个集群无法继续保证高可靠性时，系统会自动发送相关告警，提示用户进行修复。

数据导入导出工具

Redis集群构建了16384个槽位，通过计算不同key的crc16码值来决定key存放在哪个槽位，用来做键值的哈希，来保证不同的主实例负载均衡。这样不同的槽位存放着不同的值，如果两个集群的拓扑结构不相同，则其不同实例上存放着的键是不同的，此时如果需要进行数据迁移或者备份恢复是非常困难的。

FusionInsight HD提供了一个专用的数据导入导出工具，可以方便的导出Redis集群中的数据，并支持在原集群、新集群、异构集群（节点个数不同的集群）进行数据恢复。

完善的安全特性

社区redis仅提供简单的密码认证机制，其密码以明文的方式保存在配置文件当中，在企业级应用当中，这种安全机制是不够的。FusionInsight HD提供了完善的安全特性，增加了认证、鉴权、审计机制。

只有认证过的客户端才能向服务端发送或请求数据，而服务端之间（集群内部）也有认证机制，防止伪造的实例发送未认证的请求。 同时鉴权管理机制则将Redis的命令分为三类：读取型、写入型、管理型，不同的用户赋予不同的权限，避免越权的操作。

审计机制则对Redis的一些高危操作记录了详细的审计日志，比如更改集群拓扑结构、清除Redis中的数据等。

极致的性能优化

Redis已经是一个高性能的分布式缓存数据库，但原始的Redis实例部署在普通OS上，会发现存在这样一个问题：当客户端并发量提高时，尽管服务端CPU等资源并不存在瓶颈，但TPS吞吐量却无法继续提高。当组建Redis集群的时候，集群的性能也无法随着集群规模的扩大而线性提升。 FusionInsight HD在OS层面进行了多重优化，包括CPU核绑定、网卡中断队列绑定、OS系统参数优化等，从而保证了Redis的性能，特别是Redis集群性能的线性增长。

图 4-23 单实例优化前后测试

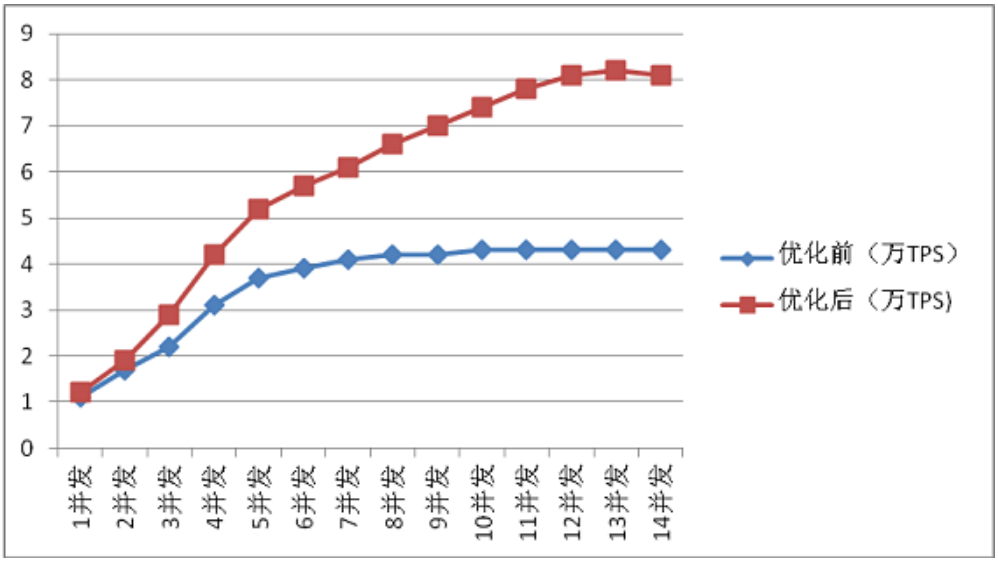
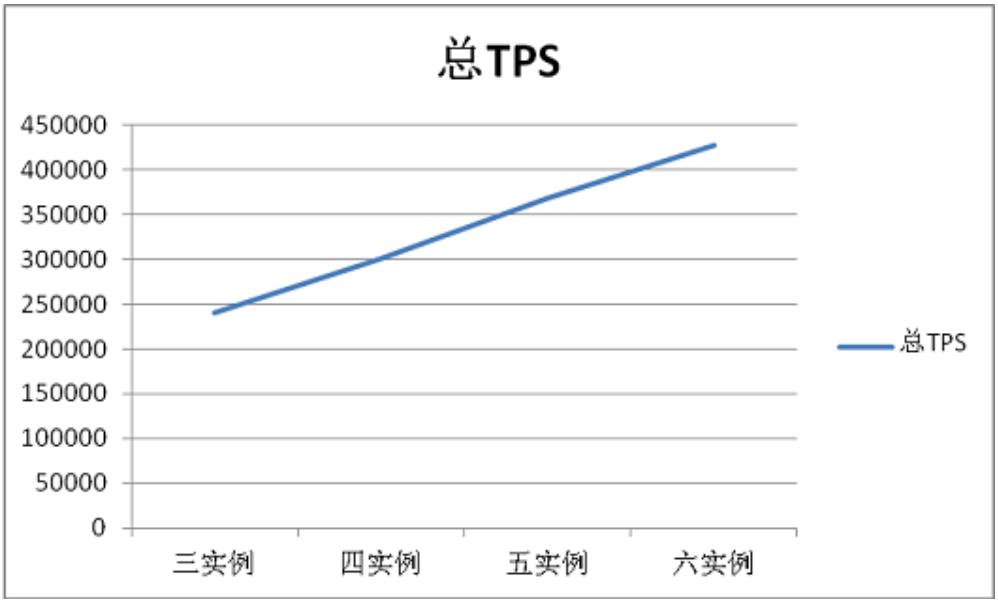


图 4-24 集群场景优化后性能线性扩展测试



增强的淘汰算法

Redis是一个缓存系统，在其内存达到配置的最大值之后，将会触发数据的淘汰。原生Redis具备LRU、Random、TTL三种淘汰策略，但是在实际业务使用中并不能很好的达到“淘汰冷数据、留下热数据”的目的。

FusionInsight HD中的Redis组件对淘汰算法做了增强，引入了Smart淘汰策略，该淘汰策略基于对key的热度统计进行淘汰，确保每次尽量只淘汰最冷的数据。在模拟的业务测试中，Smart淘汰策略的热数据命中率可以始终保持在99%以上，热数据淘汰率最高则也只有3%左右（原生LRU策略的这两个测试数据则分别是85%和35%）。由于热数据命中率的提高，从而也使得业务请求的TPS得到提高。

集群管道

redis-server支持客户端发过来的管道命令，即一次接收多个命令进行处理，这样能减少网络传输时间，极大提高redis-server的每秒请求处理数量。但是Jedis社区仅提供单实例的管道模式。华为客户端通过对Jedis进行封装，使得这种模式也能应用在集群上，并且使用方式与单实例的管道模式保持一致。

4.17 安全增强

华为FusionInsight HD作为一个海量数据管理和分析的平台，具备高安全性。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

  整个系统网络划分为2个平面，即业务平面和管理平面。两个平面采用物理隔离的方式进行部署，保证业务、管理各自网络的安全性。

  - 业务平面通过业务网络接入，主要为用户和上层用户提供业务通道，对外提供数据存取、任务提交及计算的能力。
  - 管理平面通过运维网络接入，提供系统管理和维护功能，主要用于集群的管理，对外提供集群监控、配置、审计、用户管理等服务。

- 主机安全

  用户可以根据自己业务的需要部署第三方的防病毒软件。针对操作系统和端口部分，华为提供如下安全措施。

  - 操作系统内核安全加固
  - 更新操作系统最新补丁
  - 操作系统权限控制
  - 操作系统端口管理
  - 操作系统协议与端口防攻击

- 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：FusionInsight HD通过配置主、备集群关系和需要同步的数据表，提供异地实时容灾功能。当主集群数据遭到破坏（例如，洪水、地震等），备集群可以立即接管业务。
  - 备份：FusionInsight HD支持针对OMS、HBase、HDFS、LDAP数据的备份。

- 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。

  - 软件安装时支持对软件包的sha256校验，防止软件包或者程序被篡改。
  - 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读数据的时候同会检查数据是否完整，如果发现数据不完整，尝试从其他的DataNode节点上读取数据。

- 数据保密性

  FusionInsight HD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，遵从帐户/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，FusionInsight HD使用LDAP作为帐户管理系统，并通过Kerberos对帐户信息进行安全认证。
- 提供单点登录，统一了FusionInsight HD系统用户和组件用户的管理及认证。
- 对登录FusionInsight Manager的用户进行审计。
- 提供统一证书管理功能，并基于界面进行整个集群的证书配置和更换，解决了用户针对单个组件需要依次更换证书的繁琐操作。

4.18 可靠性增强

FusionInsight HD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

- 所有组件的管理节点均实现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。

  华为FusionInsight HD产品对所有业务组件的管理节点都提供了类似的双机机制，包括OMS Server、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。

- 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。

- HBase集群异地容灾

  HBase集群通过实时异地容灾来提高HBase集群系统的可靠性，是业界第一个实现1000公里以上的大数据集群HBase容灾系统。主备容灾系统之间相互进行健康状态监视和功能切换，当一处系统因意外(如火灾、洪水、地震、人为蓄意破坏等)停止工作时，整个应用系统可以切换到另一处，使得该系统功能可以继续正常工作。

  HBase集群容灾对外还提供了基础的运维工具，包含灾备关系维护、重建、数据校验、数据同步进展查看等功能。

- 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，FusionInsight HD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

    FusionInsight HD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HBase、OMSServer、LDAPServer、DBService的数据以及ESN编码。

- 手动备份

  在系统进行扩容、升级、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

  FusionInsight HD针对操作系统提供了如下监控措施：

  - 支持开启硬件看门狗功能。
  - 支持对操作系统内核参数进行微调，在操作系统出现致命异常，如内存耗尽、非法地址访问、内核死锁、调度器失效时，重启操作系统，恢复业务。
  - 周期采集操作系统运行状况数据，包括处理器状态、内存状态、硬盘状态、网络状态等。

- 进程健康状态监控

  FusionInsight HD在各节点上部署了代理进程NodeAgent，负责监控业务实例的状态以及业务实例进程的健康指标信息

- 硬盘故障的自动处理

  FusionInsight HD对开源版本进行了增强，可以监控各节点上的硬盘状态，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），也会采取措施，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，从而大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RAID组的配置

  FusionInsight HD建议按照实际业务需要，合理规划节点的硬盘资源，以提高系统对硬盘故障的抵御能力。

  - 各节点的操作系统，建议安装在两块硬盘做成的RAID1上，以保障系统盘的稳定。
  - 如果条件允许，管理节点关键进程使用的硬盘（如HDFS NameNode、数据库、ZooKeeper等）尽量做成RAID1，以保证元数据的可靠性。
  - 数据盘（HDFS DataNode）可以做成单盘RAID0（即每个RAID0组内只有1块硬盘），或者做成NonRaid。

数据可靠性

FusionInsight HD通过对节点硬件（特别是硬盘）、操作系统、进程的监控，及时发现相关部件的异常状况，缩短了对应部件的故障检测时间和修复时间，从而提高了系统整体的数据持久度。

5 系统规格

FusionInsight HD的系统规格如表5-1所示。

表 5-1 系统规格

指标名称	规格	说明
系统最大节点数	5000个	通用X86服务器（不限于华为服务器）
单Storm集群节点数	128个	通用X86服务器（不限于华为服务器）
单Kafka集群节点数	64个	通用X86服务器（不限于华为服务器）
单Redis集群实例数	512个	Redis进程数
Redis集群数	512个	Redis集群的最大数
单Solr集群实例数	500个	Solr进程数
租户数量	512个	租户的最大数