10

Hadoop生态系统

Hadoop生态系统

TOC

10.1 Hadoop总体架构

系统架构：

- 部署在低成本的Intel/Linux硬件平台上
- 由多台装有Intel x86处理器的服务器或PC机组成
- 通过高速局域网构成一个计算集群
- 各个节点上运行Linux操作系统

三大主要模式：

- 单机模式（standalone mode）
- 虚拟分布模式（pseudo-distributed mode）
- 完全分布模式（completely distributed Mode）

10.1 Hadoop总体架构

集群配置：

硬件配置：

- NameNode（执行作业调度、资源调配、系统监控等任务）
- DataNode（承担具体的数据计算任务）

软件配置：

- Linux O/S
- JDK 1.6以上版本
- SSH（Security Shell）安全协议

网络配置：

- NameNode到机架（Rack）的网络连接
- 机架内部的DataNode之间的网络连接

10.1 Hadoop总体架构

集群软件配置：

主节点运行的程序或进程：

- 主节点程序Namenode
- Jobtracker 守护进程
- 管理集群所用的Hadoop 工具程序和集群监控浏览器

从节点运行的程序：

- 从节点程序Datanode
- 任务管理进程Tasktracker

区别：

- 主节点程序提供 Hadoop 集群管理、协调和资源调度功能
- 从节点程序主要实现 Hadoop 文件系统（HDFS）存储功能和节点数据处理功能。

10.1 Hadoop总体架构

Hadoop软件架构：

组成：

- 基于HDFS/HBase的数据存储系统
- 基于YARN/Zookeeper的管理调度系统
- 支持不同计算模式的处理引擎

10.1 Hadoop总体架构

数据存储系统

组成：

- 分布式文件系统HDFS（Hadoop Distributed File System）
- 分布式非关系型数据库Hbase
- 数据仓库及数据分析工具Hive和Pig
- 用于数据采集、转移和汇总的工具Sqoop和Flume。

HDFS文件系统构成了Hadoop数据存储体系的基础

10.1 Hadoop总体架构

管理调度系统：

Zookeeper：提供分布式协调服务管理

Oozie：负责作业调度

Ambari：提供集群配置、管理和监控功能

Chukwa：大型集群监控系统

YARN：集群资源调度管理系统

10.2 HDFS文件系统

分布式文件系统：

结构：

- 物理存储资源和对象分散存储在通过网络相连的远程节点上
- 主控服务器（也称元数据服务器）：负责管理命名空间和文件目录，
- 远程数据服务器（也称存储服务器）节点：存储实际文件数据

特点

- 透明性
- 高可用性
- 支持并发访问
- 可扩展性
- 安全性

10.2 HDFS文件系统

分布式文件系统：

物理存储资源和对象分散存储在通过网络相连的远程节点上

主控服务器（也称元数据服务器）：负责管理命名空间和文件目录，

远程数据服务器（也称存储服务器）节点：存储实际文件数据

特点

- 透明性
- 高可用性
- 支持并发访问
- 可扩展性
- 安全性

10.2 HDFS文件系统

HDFS体系结构：

唯一主节点：运行NameNode，JobTracker，Zookeeper，Hmaster等负责集群管理、资源配置、作业调度的程序

多个从节点（dataNode）：承担数据存储及计算任务。

客户端（Client）：用于支持客户操作HDFS

10.2 HDFS文件系统

HDFS架构：

Master/Slave架构，集群中只设置一个主节点

优：

- 简化了系统设计
- 元数据管理和资源调配更容易

劣：

- 命名空间的限制
- 性能的瓶颈
- 单点失效（SPOF）问题：

10.2 HDFS文件系统

HDFS存储结构：

以块（block）为基本单位存储文件

每个文件被划分成64MB大小的多个blocks，属于同一个文件的blocks分散存储在不同DataNode上；

出于系统容错需要，每一个block有多个副本（replica），存储在不同的DataNode上；

每个DataNode上的数据存储在本地的Linux文件系统中。

10.2 HDFS文件系统

HDFS存储结构优势：

有利于大规模文件存储

适合数据备份

系统设计简化

10.2 HDFS文件系统

HDFS命名空间管理：

命名空间包括目录、文件和块

文件 -> block -> 节点的映射关系作为元数据存储在Namenode上

整个HDFS集群只有一个命名空间，由唯一的一个名称节点负责对命名空间进行管理

HDFS使用的是传统的分级文件体系

NameNode进程使用FsImage和EditLog对命名空间进行管理。

10.2 HDFS文件系统

FsImage：

存储和管理内容：

- 文件系统目录树
- 目录树中所有文件和文件夹的元数据

由名称节点进程把文件 -> block -> 节点映射关系表装载并保留在内存中。

EditLog：

是NameNode启动后对文件系统改动操作的记录

10.2 HDFS文件系统

第二名称节点:

作用:

- 保存名称节点对HDFS元数据信息的备份
- 减少名称节点重启的时间

一般独立部署在一台机器上

工作流程:

- Roll edits
- Retrieve FsImage and edits from NameNode
- Merge
- Transfer checkpoint to NameNode
- Roll again:

10.2 HDFS文件系统

HDFS文件读写机制

主要访问方式：

- HDFS shell命令
- HDFS Java API

10.2 HDFS文件系统

HDFS读文件流程（以JAVA为例）

打开文件

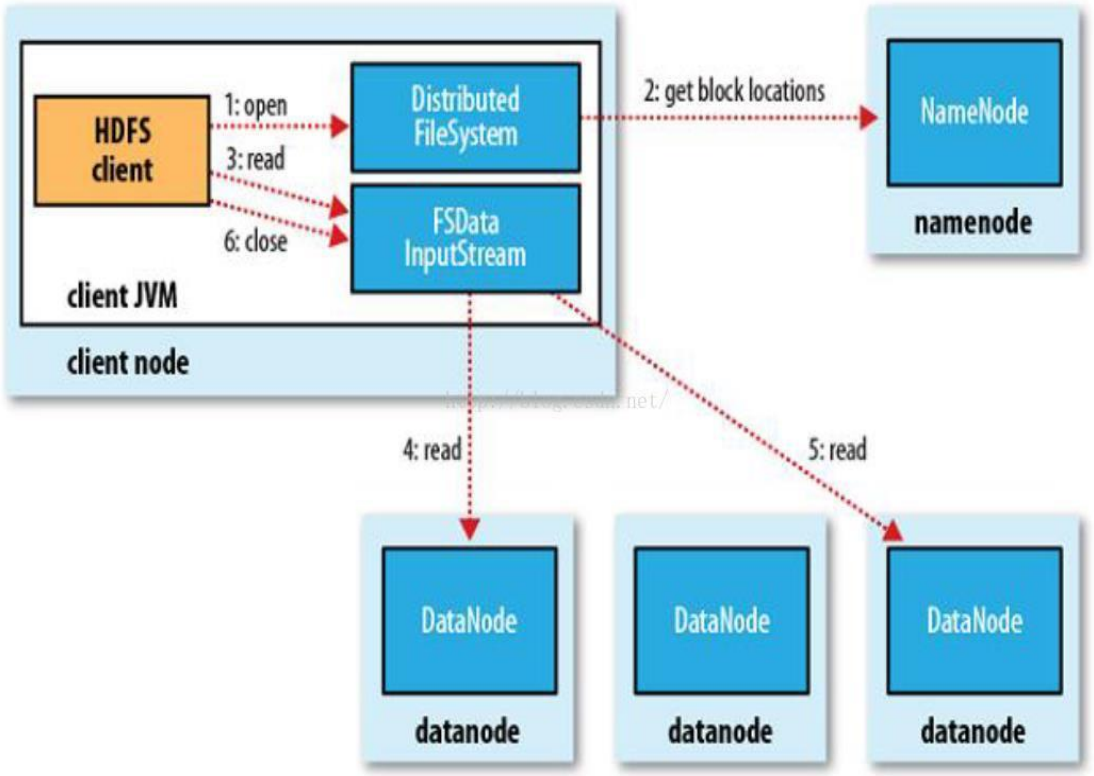
获取块信息

读取请求

读取数据

读取下一个数据块

关闭文件



10.2 HDFS文件系统

HDFS写文件流程（以JAVA为例）

创建文件

建立文件元数据

写入请求

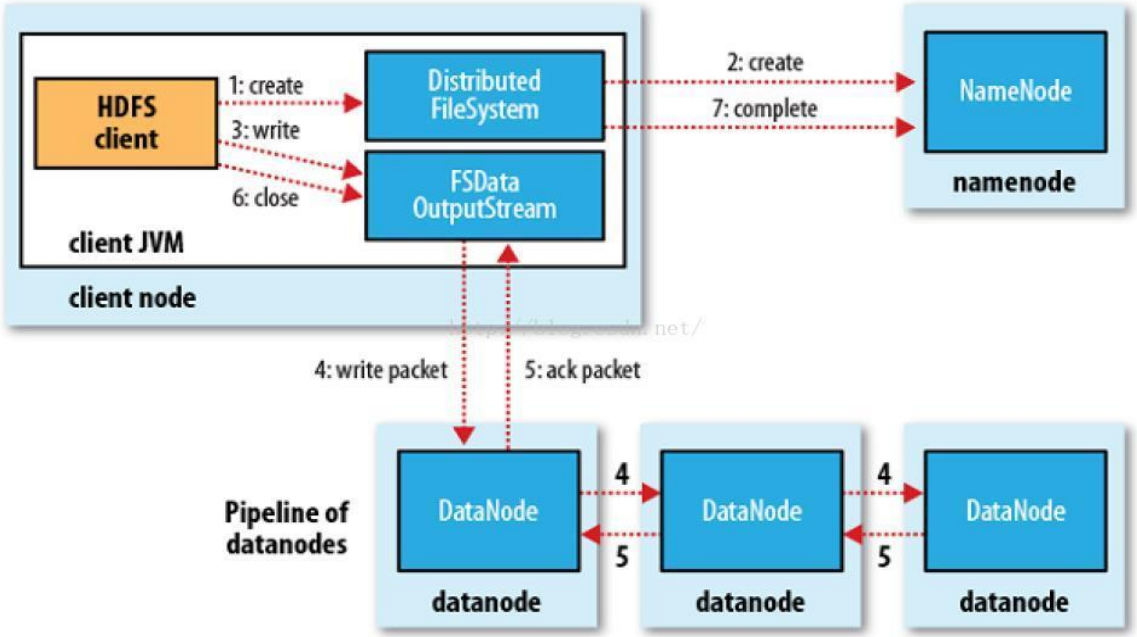
写入数据包

接收确认包

关闭文件

结束过程

通知名称节点关闭文件



10.2 HDFS文件系统

HDFS数据容错与回复机制

多副本方式进行冗余存储

- 加快数据传输速度
- 容易检查数据错误
- 保证数据可用性

机架感知副本存放策略

- 改进数据的可靠性、可用性和网络宽带的利用率
- 防止某一机架失效时数据丢失
- 利用机架内的高带宽特性提高数据读取速度

错误检测和恢复机制

- 包括NameNode检测、DataNode检测和数据错误检测

10.2 HDFS文件系统

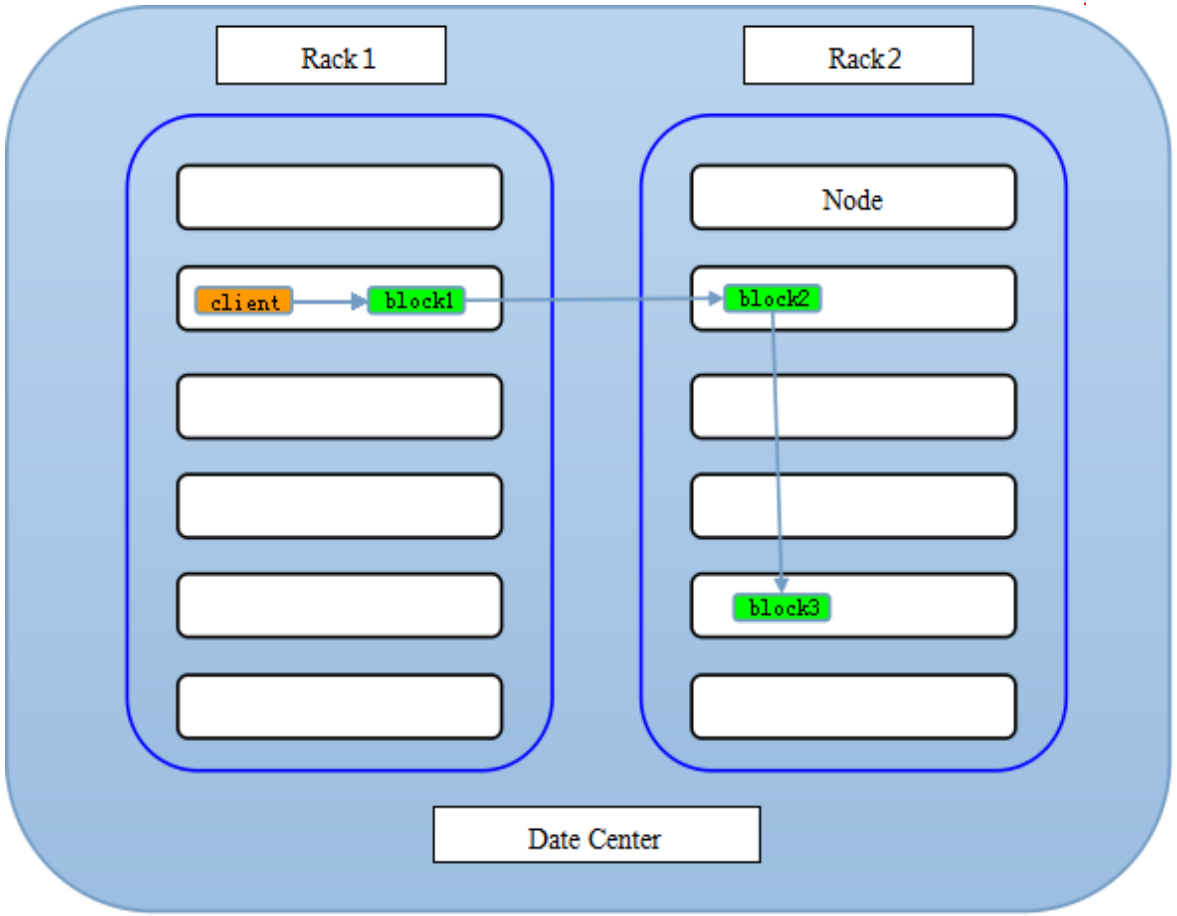
机架感知副本存放

存放流程

block1放到与客户端同一机架的一个节点

block2放到block1所在机架之外的节点

block3放在与block2同一机架的另一节点



10.2 HDFS文件系统

机架感知副本存放策略

读取流程

HDFS提供了一个API可以确定某一数据节点所属的机架ID

客户端从名称节点获得不同副本的存放位置列表

调用API确定这些数据节点所属的机架ID

发现ID匹配：优先读取该数据节点存放的副本

没有发现：随机选择一个副本读取数据

10.2 HDFS文件系统

HDFS文件错误检测和恢复机制

NameNode检测：第二名称节点

DataNode检测：心跳检测

数据错误检测：CRC循环校验

10.2 HDFS文件系统

DataNode检测：心跳检测机制

DataNode周期性的向集群NameNode发送心跳包和块报告

出现情况	应对
规定时间内未收到心跳报告	将该DataNode标记为失效
数据块副本的数目低于设定值	启动数据冗余复制，为该数据块生成新的副本，放置在另外节点上
数据副本损坏、DataNode上的磁盘错误或者复制因子增大	触发复制副本进程

10.3 分布式存储架构

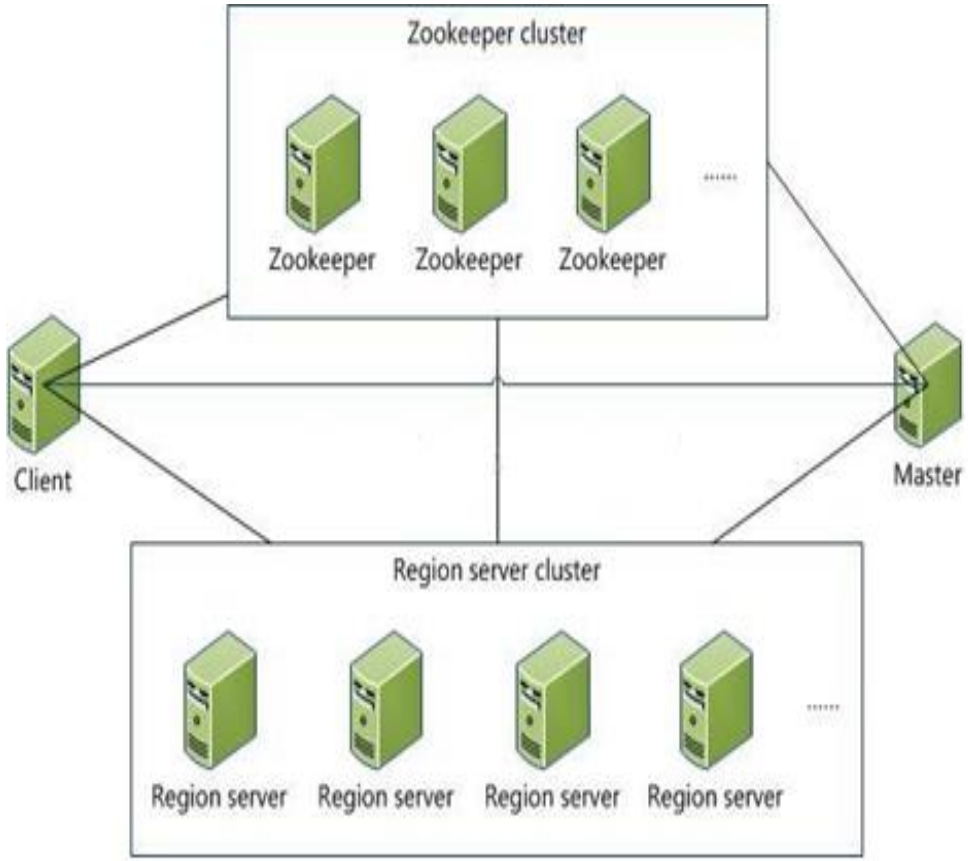
本节以Hbase为例讲述分布式存储架构

Hbase集群部署

物理部署：Hadoop集群

软件部署：四大组件

- Master
- Region Server
- Zookeeper
- Client



10.3 分布式存储架构

Hbase系统架构

Hadoop基础平台提供了物理结构

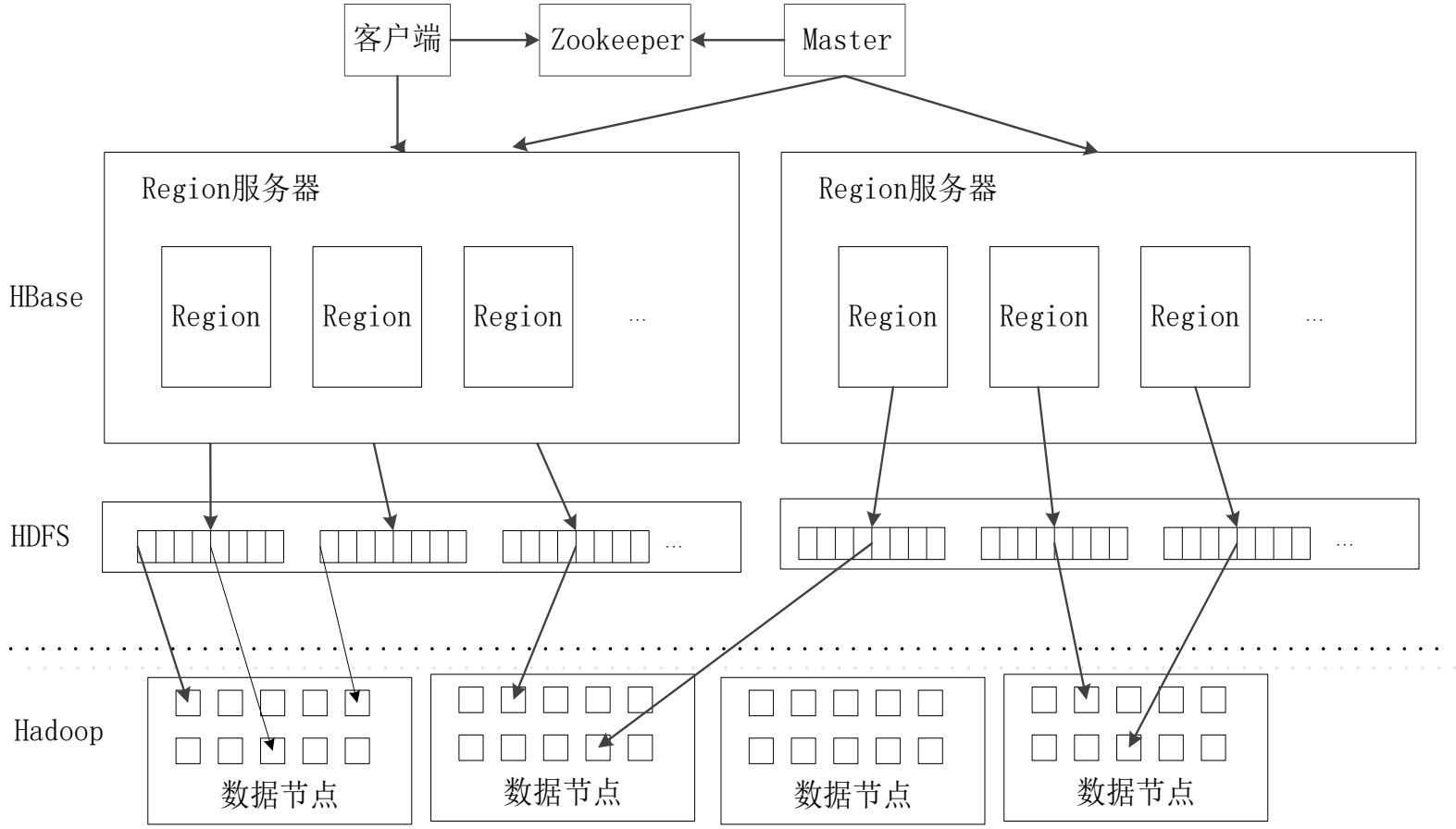
HDFS提供了HBase的底层数据存储结构

Master节点管理着整个HBase集群

Region Server管理多个regions并提供数据访问服务

Zoopkeeper负责分布式协调服务，客户端提供了数据库访问接口。

10.3 分布式存储架构


客户端
Zookeeper
Master
Region服务器
Region服务器
HBase
Region
Region
Region
...
Region
Region
Region
...
HDFS
...
...
Hadoop
数据节点
数据节点
数据节点
数据节点

10.3 分布式存储架构

Hbase相关基本概念

Region

Store

HFile

10.3 分布式存储架构

Region

是将数据表按照RowKey划分形成的子表

是数据表在集群中存储的最小单位

可以被分配到某一个Region Server进行存储管理

各个Region Server存放的Region数目大致相同，以达到负载均衡的目的。

Region内部包含一个HLog日志和多个Store，数据实际上是存储在Store单元中。

10.3 分布式存储架构

Store

Region内部按照列簇分为不同的Store

每个Store由一个memStore和多个StoreFile组成

memStore是内存中的一个缓存区

StoreFile是写到硬盘上的数据文件

数据首先会放入MemStore中，当MemStore满了以后会清空形成一个新StoreFile

检索数据时，先在memStore找，然后找StoreFile

10.3 分布式存储架构

Store

compact操作：

- 当StoreFile文件数量增长到一定阈值时触发
- 将多个StoreFile合并成一个StoreFile
- 在合并过程中会进行StoreFile版本合并和数据删除。

split操作：

- 当单个StoreFile大小超过一定阈值后触发
- 把当前的Region分裂成2个子Regions
- 子Region会被Master分配到相应的Region Server上
- 是HBase提供的负载均衡机制

10.3 分布式存储架构

HFile

StoreFile包含的一个HFile文件

是Hadoop的二进制格式文件

StoreFile是HFile的轻量级包装，数据最终是以HFile的形式存储在Hadoop平台上

采用一个简单的byte数组存储数据的每个KeyValue对

这个byte数组里面包含了很多项，有固定的格式，每项有具体的含义。

10.3 分布式存储架构

HFile

Byte数组组成

Data Block 段：用来保存表中的数据

Meta Block段 (可选)：保存用户自定义的Key-Value对

File Info段：HFile的元信息

Data Block Index段：Data Block的索引

Meta Block Index段 (可选)：Meta Block的索引

Trailer段：保存每一段的偏移量

10.3 分布式存储架构

Hbase数据模型与存储模式

HBase表特性：

- 面向列的、稀疏的、分布式的、持久化存储的多维排序映射表

Hbase表索引：

- 行关键字、列簇名、列关键字及时间戳

Hbase表值形式：

- 一个未经解析的byte数组

10.3 分布式存储架构

Hbase数据模型与存储模式

Hbase数据模型：

- 以表的形式存储数据
- 表由行和列族组成
- 一个表可包含若干个列族
- 一个列族内可用列限定符来标志不同的列
- 存于表中单元的数据尚需打上时间戳

10.3 分布式存储架构

Hbase数据模型与存储模式

Hbase数据模型基本元素：

- 表
- 行键
- 列族
- 单元格
- 时间戳

10.3 分布式存储架构

Hbase数据模型与存储模式

Hbase存储逻辑视图：

- 一个三元组（行键，列族:列限制符，时间戳）可以唯一地确定存储在单元（Cell）中的数据
- Key是一个三元组（行键，列族:列限制符，时间戳）
- Value就是这个三元组定位的数据值

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor	ColumnFamily people
"com.cnn.www"	t9		anchor:annsi.com ="CNN"	
"com.cnn.www"	t8		anchor:my.look.c a="CNN.com"	
"com.cnn.www"	t6	contents:html= "<html>..."		
"com.cnn.www"	t5	Contents:html= "<html>..."		
"com.cnn.www"	t3	contents:html= "<html>..."		

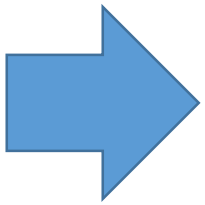
10.3 分布式存储架构

Hbase数据模型与存储模式

Hbase存储物理视图：

- 一个列族对应生成一个Region

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor	ColumnFamily people
"com.cnn.www"	t9		anchor:annsi.com ="CNN"	
"com.cnn.www"	t8		anchor:my.look.c a="CNN.com"	
"com.cnn.www"	t6	contents:html= "<html>..."		
"com.cnn.www"	t5	Contents:html= "<html>..."		
"com.cnn.www"	t3	contents:html= "<html>..."		



行键	时间戳	列族 contents
"com.cnn.www"	t6	contents:html="<html>..."
	t5	contents:html="<html>..."
	t3	contents:html="<html>..."

行键	时间戳	列族 anchor
"com.cnn.www"	t9	anchor:cnnsi.com="CNN"
	t8	contents:my.lool.ca="CNN.com"

10.3 分布式存储架构

Hbase数据模型与存储模式

Hbase物理存储：

- 表划分出的列族对应着物理存储区的Region
- 列族所包含的列对应着的存储区Region所包含的Store
- 当增大到一个阀值的时候，Region就会等分成两个新的Region

10.3 分布式存储架构

Hbase寻址机制

三层机构：

- Zookeeper文件
- -ROOT-表
- .META.表

客户端从Zookeeper获得Region的存储位置信息后，直接在Region Server上读写数据

流程：Zookeeper→-ROOT-表→.META.表→找到存放用户数据的Region Server位置

10.3 分布式存储架构

Hbase扫描读取数据

所有的存储文件被划分成若干个存储块

存储块在get或scan操作时会加载到内存中

HBase顺序地读取一个数据块到内存缓存中

再读取相邻数据时从内存中读取而不需要读磁盘

10.3 分布式存储架构

Hbase写数据

- Client向Region Server提交写数据请求；
- Region Server找到目标Region；Region检查数据是否schema一致；
- 如果客户端没有指定版本，则获取当前系统时间作为数据版本；
- 将数据更新写入HLog（WAL），只有HLog写入完成之后，commit()才返回给客户端；
- 将数据更新写入MemStore；
- 判断MemStore的是否需要flush为StoreFile，若是，则flush生成一个新StoreFile;
- StoreFile数目增长到一定阈值，触发compact合并操作，多个StoreFile合并成一个StoreFile，同时进行版本合并和数据删除；
- 若单个StoreFile大小超过一定阈值，触发split操作，把当前Region拆分成2个子Region，原来的Region会下线，新分出的2个子Region会被Master重新分配到相应的Region Server上

10.3 分布式存储架构

Hbase更新表

首先写入HLog和MemStore

MemStore中的数据是排序的

当MemStore累计到一定阈值时：

- 创建一个新的MemStore，
- 将老的MemStore添加到flush队列，由单独的线程刷写到磁盘上，成为一个新StoreFile
- 系统在HLog中记录一个检查点，表示这个时刻前的变更已持久化

10.3 分布式存储架构

Hbase预防数据丢失

每个Region服务器都有一个自己的HLog 文件

每次启动都检查HLog文件，确认最近一次执行缓存刷新操作之后是否发生新的写入操作

发现更新时：

- 写入MemStore
- 刷写到StoreFile
- 删除旧的Hlog文件，开始为用户提供服务

10.3 分布式存储架构

StoreFile合并与分裂

合并：

- 时机：当一个Store中的StoreFile达到一定的阈值时
- 操作：将同一个key的修改合并到一起，形成一个大的StoreFile

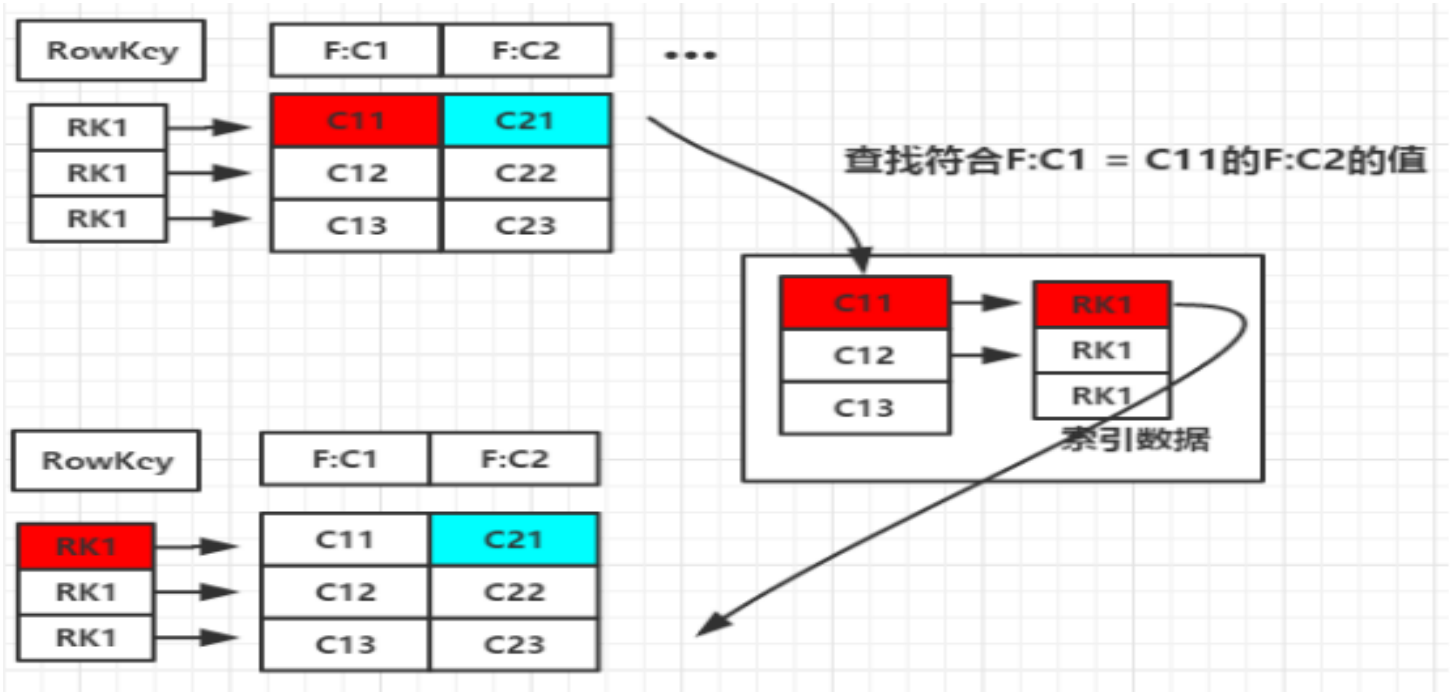
分裂：

- 时机：当StoreFile的大小达到一定阈值后
- 操作：等分为两个StoreFile。

10.3 分布式存储架构

Hbase索引与检索

机制：二次索引表机制



10.3 分布式存储架构

二次索引表机制

关键：建立主表列到RowKey的逆向映射关系

实现技术：

- 表索引
  - 主表的索引列值为索引表的RowKey，
  - 主表的RowKey做为索引表的Qualifier或Value
- 列索引
  - 增加一个单独列族存储索引值
  - 主表的用户数据列值做为索引列族的Qualifier
  - 用户数据Qualifier做为索引列族的列值

10.4 Hadoop资源管理与作业调度

实现方案：三大组件

- Zookeeper提供分布式协同服务
- Oozie 提供作业调度和工作流执行
- YARN 提供集群资源管理服务

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

提供服务

- 统一命名服务
- 应用配置管理
- 分布式锁服务
- 分布式消息队列

架构：主从架构

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

Zookeeper服务由一组Server节点组成

- 每个节点上运行一个Zookeeper程序

每个server维护内容：

- 自身的内存状态镜像、持久化存储的事务日志和快照

ZooKeeper集群的数量一般为奇数

有过半Server可用，整个系统即保持可用性。

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

节点角色

Leader：负责进行 投票的发起和决议，更新系统状态，管理系统元数据；

Follower：用于接受客户端服务请求并向客户端返回结果，在选举Leader过程中参与投票；

Observer：可以接受客户端请求并将写数据请求转发给Leader，但Observer 不参加投票过程，只同步Leader的状态数据。Observer的目的是为了扩展系统， 提高读取速度。

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

失效处理机制

- Zookeeper作出快速响应
- 消息层基于Fast Paxos算法重新推举一个Leader，继续作为协调服务中心处理客户端的写数据请求，并将ZooKeeper协同数据的变更同步（广播方式）到其他的Follower节点

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

业务流程

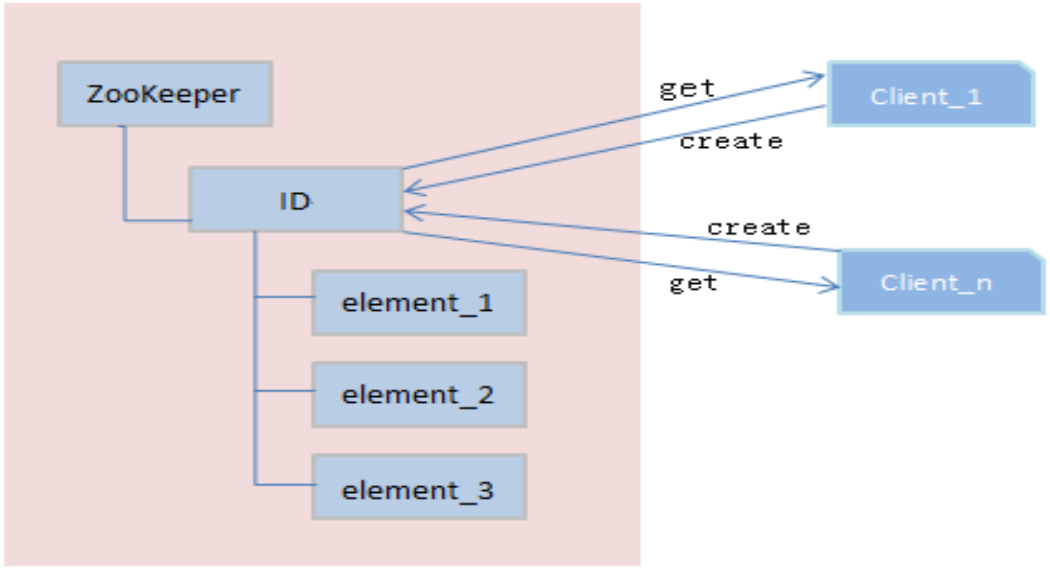
- 客户端Client连接到Follower发出写数据请求
- 请求发送到Leader节点
- Leader完成元数据更新
- Leader上的数据同步更新到其他Follower节点

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

统一命名服务

- 把各种服务名称、地址、及目录信息存放在分层结构中供需要时读取
- 提供一个分布式序列号生成器
- 流程



10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

配置管理服务

- 发布（publish）和订阅（watch）模式
- Zookeeper将集群节点配置信息保存在某个目录节点中，然后让所有需要获得配置修改信息的Client节点都监听（watch）配置信息状态，一旦配置信息发生变化，每台Client机器就会收到Zookeeper的通知，然后从Zookeeper 获取新的配置信息同步更新到本地系统。当集群中某些节点的系统信息发生变化时，它可以主动推送给Zookeeper，而Zookeeper会通知那些对该信息感兴趣的节点，让它们去Zookeeper处获得更新数据，这即是publish模式。

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

分布锁的实现

独占锁就是所有试图获取这个锁的客户端最终只有一个成功获得这把锁。通常的做法是把Zookeeper上的一个znode看作是一把锁，通过create znode的方式来实现。所有客户端都去尝试创建 /distribute_lock 节点，最终成功的那个客户端也即拥有了这把锁。

控制时序锁就是所有试图来获取这个锁的客户端，最终都会被安排执行，只是需要有一个全局排序。做法和上面基本类似，只是这里 /distribute_lock 已预先存在，排队的客户端在它下面依次创建临时节点，而Zookeeper则保持一份顺序表,保证创建的子节点的时序性，从而也形成了每个客户端的全局时序。

10.4 Hadoop资源管理与作业调度

分布式协同管理组件Zookeeper

分布式消息队列

同步队列：当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达；

FIFO队列：按照FIFO方式操作入队和出队，例如生产者和消费者模型。

10.4 Hadoop资源管理与作业调度

作业调度与工作流引擎Oozie

核心功能：

- 工作流：定义作业任务的拓扑和执行逻辑
- 协调器：负责工作流的关联和触发分布式消息队列

工作流包括：

- 控制流节点：定义工作流的开始和结束，控制执行路径
- 动作节点：支持不同任务类型

10.4 Hadoop资源管理与作业调度

作业调度与工作流引擎Oozie

工作流流节点：

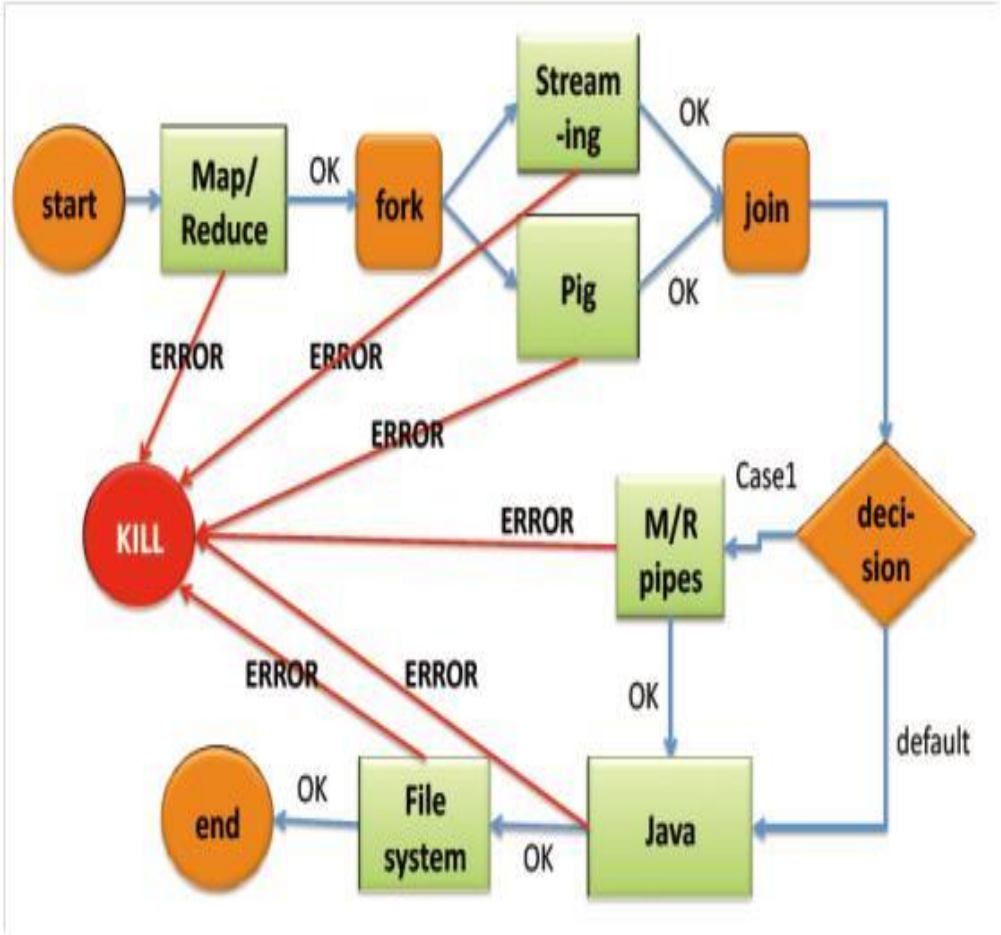
启动控制节点

末端控制节点

停止控制节点

决策控制节点

分支-联接控制节点



10.4 Hadoop资源管理与作业调度

集群资源管理框架YARN

优势：

- 允许多个应用程序运行在一个集群上，并将资源按需分配给它们，这大大提高了集群资源利用率
- YARN允许各类短作业和长服务混合部署在一个集群中，并提供了容错、资源隔离及负载均衡等方面的支持，这大大简化了作业和服务的部署和管理成本，强化了对应用程序的支持

10.4 Hadoop资源管理与作业调度

集群资源管理框架YARN

体系架构——Master/Slave架构：

- Master为YARN的Resource Manager
- Slave为NodeManager
- Application Master
- Container
- YARN Client

10.4 Hadoop资源管理与作业调度

集群资源管理框架YARN

部署方式：

- Resource Manager：部署并运行在NameNode上
- Node Manager：部署在每个DataNode上，作为Resource Manager的节点代理；
- 每个DataNode都包含一个或多个多个Container用于资源调度
- 每一个提交给Hadoop集群的Application都有一个Application Master与之对应，运行在某个DataNode上

10.4 Hadoop资源管理与作业调度

YARN资源调度模型：抽象资源模型

定义

- 不再把物理资源作为调度单位
- 把物理资源映射到抽象资源单位Container
- 基于抽象资源单位进行资源分配调度

组成

- 两层调度框架
- 基于资源预留的调度策略