



# Lecture 12 Hadoop生态系统

- Hadoop总体架构
- HDFS文件系统



## 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

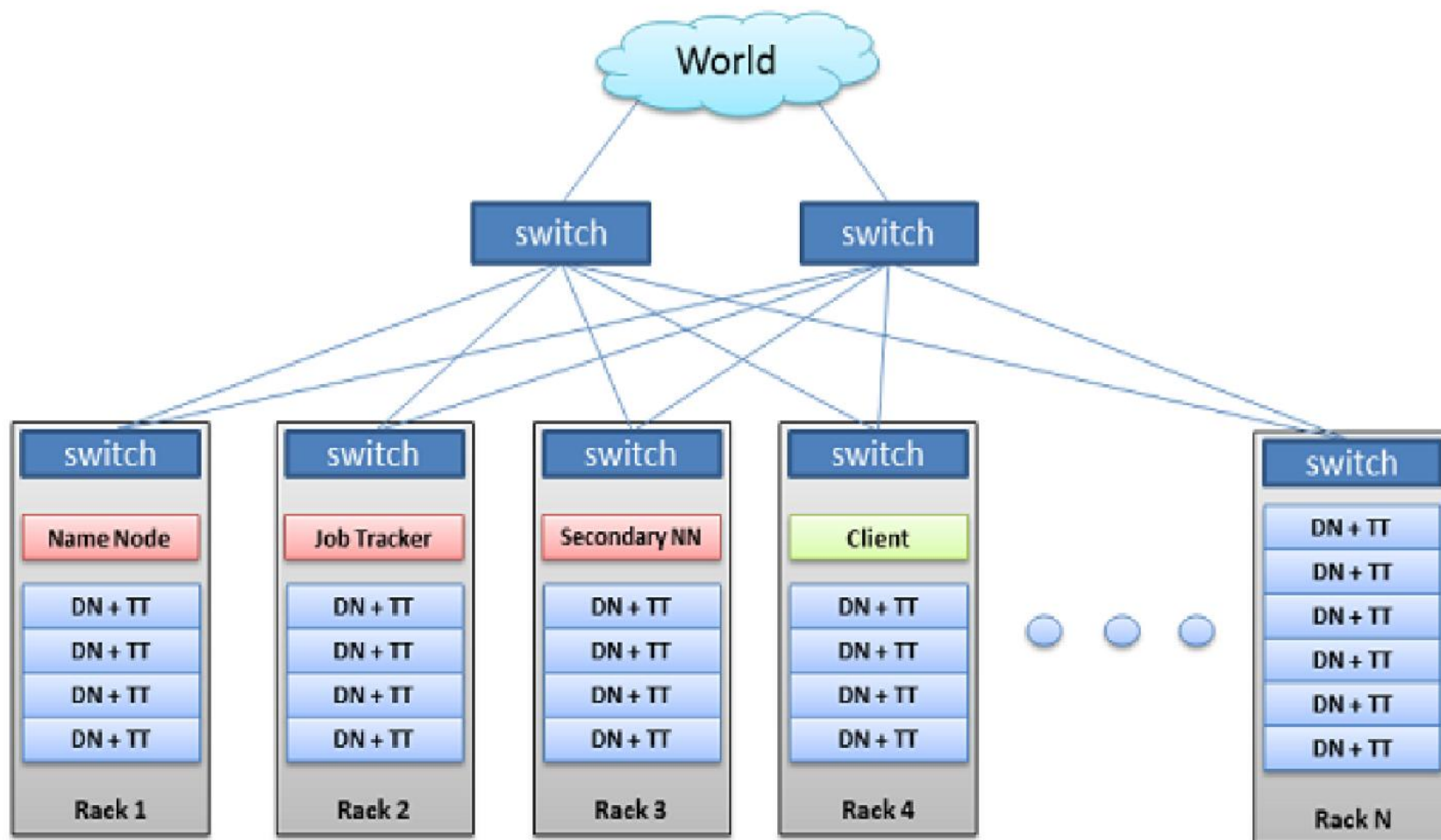
## 大数据中心机房



知乎 @奔骝科技



# 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



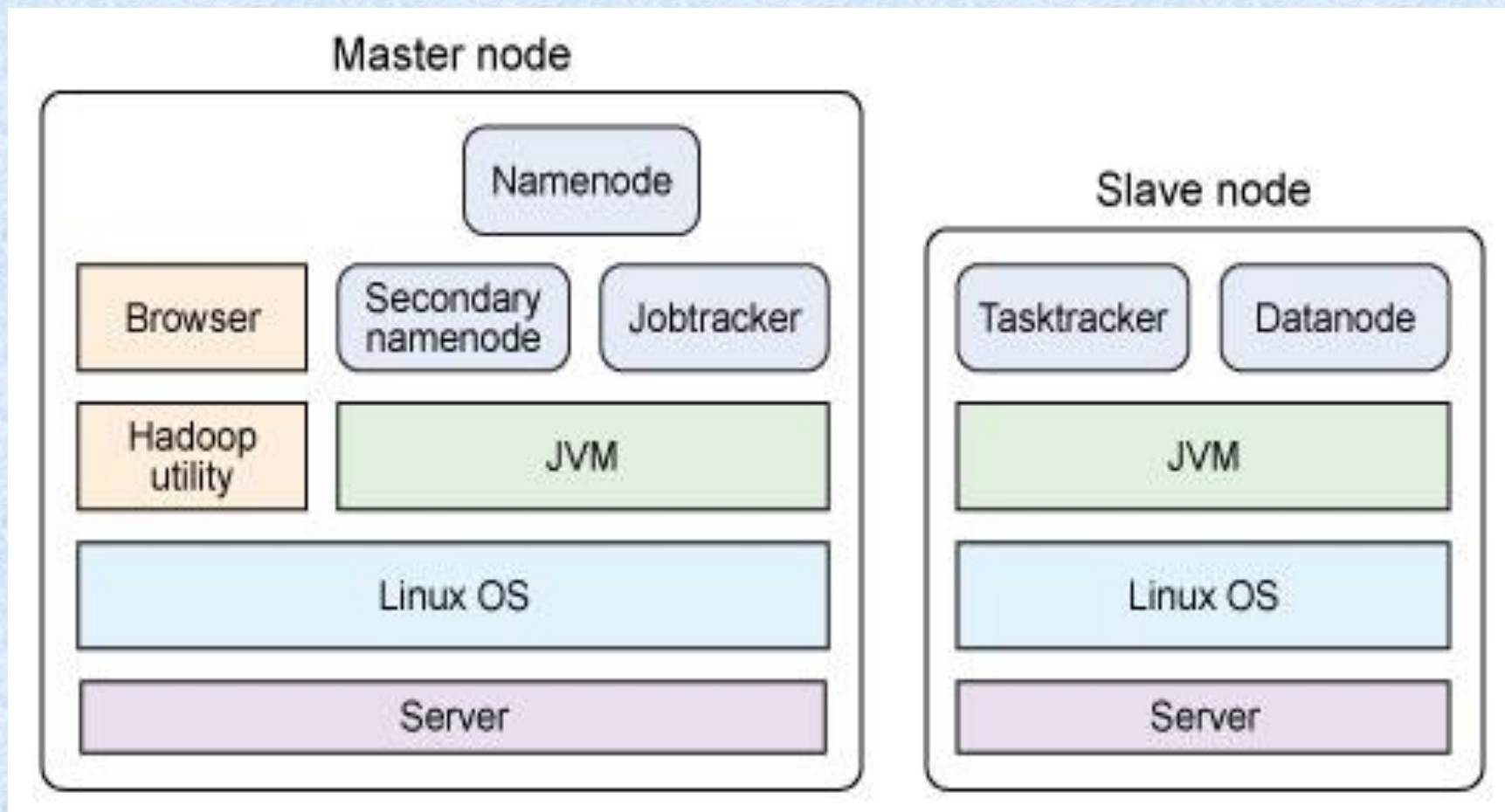
## 集群软件配置

- 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

## 节点软件部署







## Hadoop 生态系统

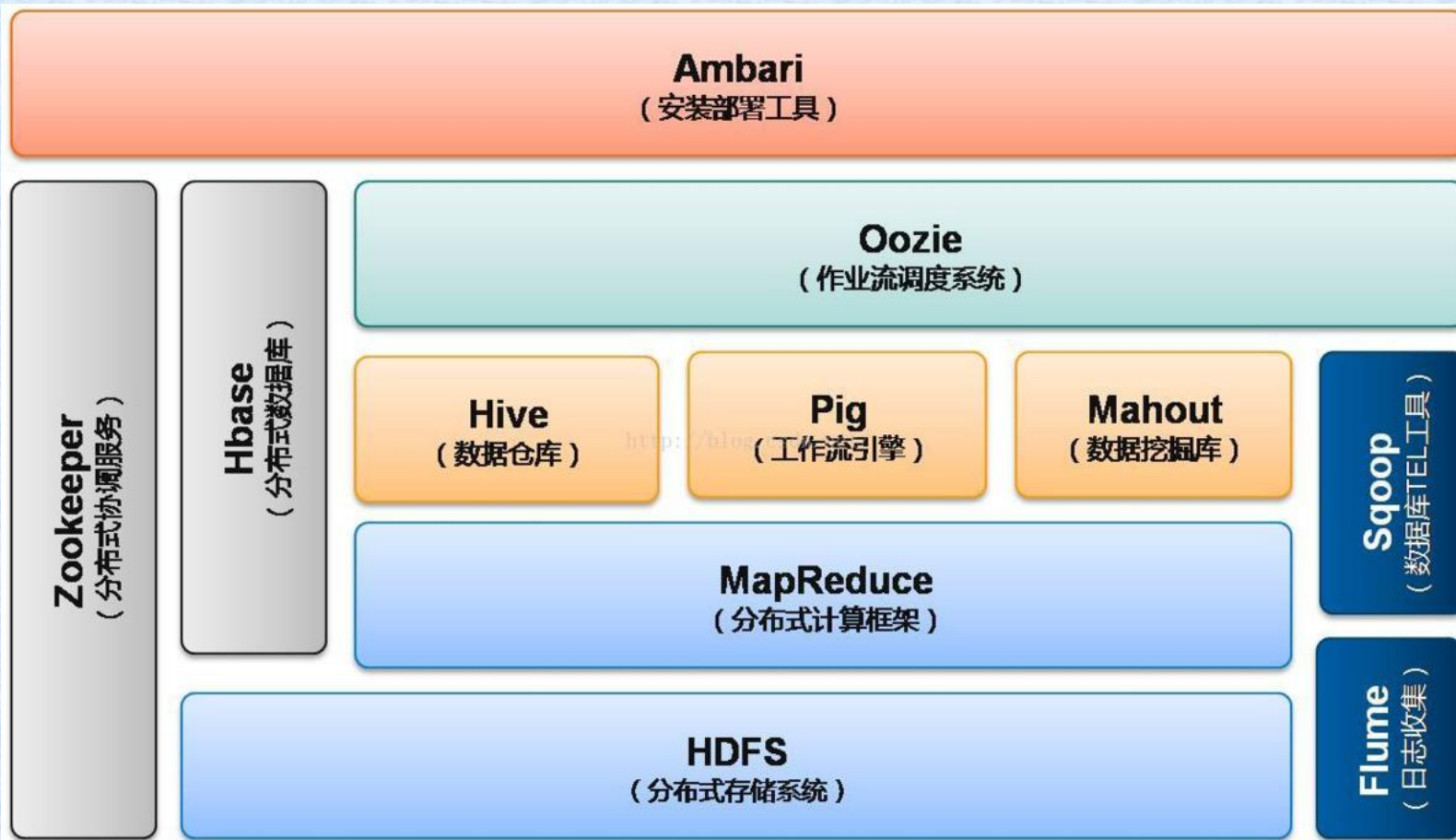
- 基于HDFS/HBase的数据存储系统
- 基于YARN/Zookeeper的管理调度系统
- 支持不同计算模式的处理引擎





# 10.1 Hadoop总体架构

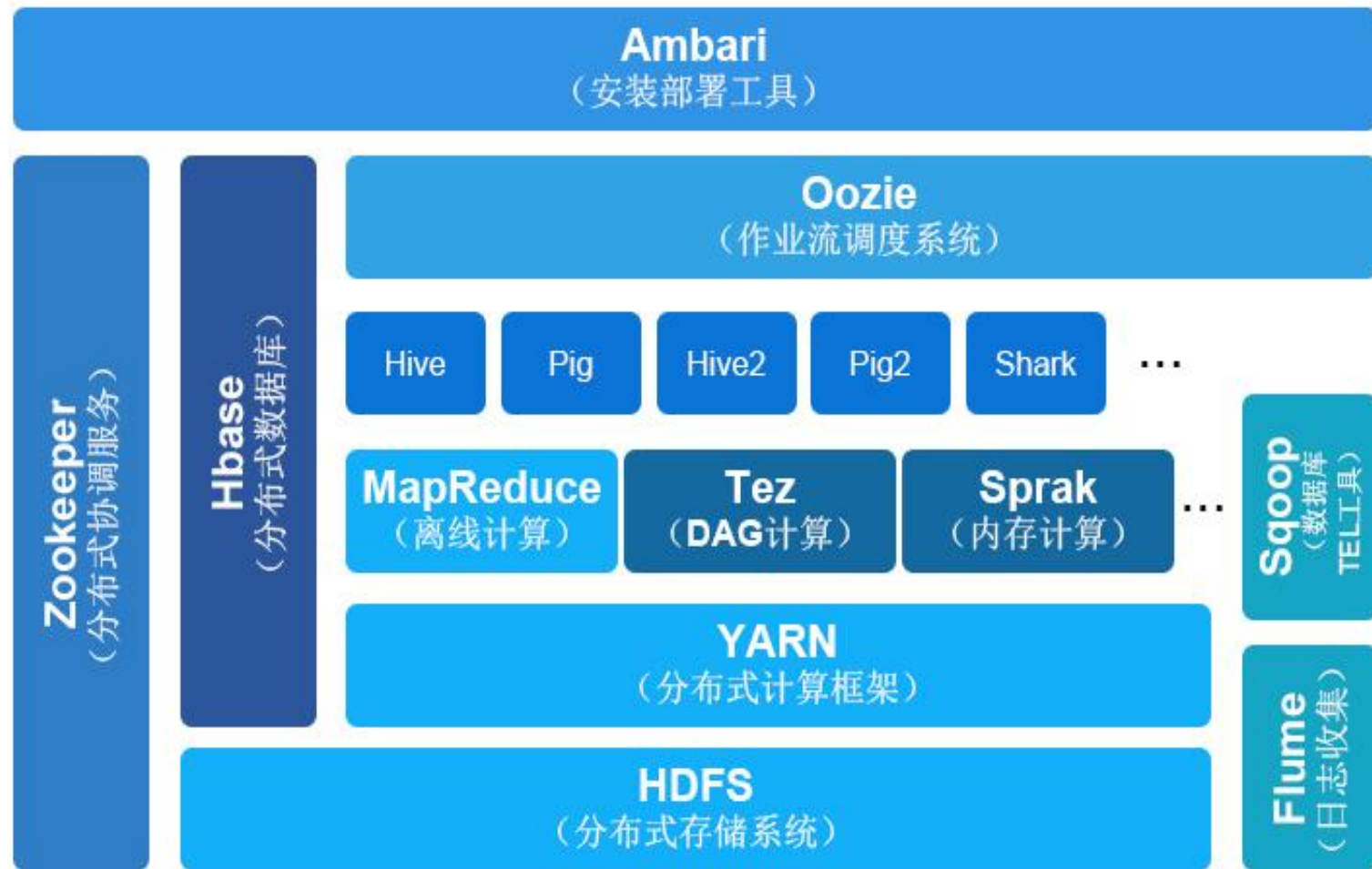
## Hadoop 1.0 生态体系





# 10.1 Hadoop总体架构

## Hadoop 2.0 生态体系





## 10.1 Hadoop总体架构

**HDFS:** Hadoop Distributed File System，分布式文件系统，具有高容错，高吞吐等特性，是常用的分布式文件存储

**MR (MapReduce):** 分布式批处理计算模型，程序人员只需在Mapper、Reducer中编写业务逻辑，然后直接交由框架进行分布式计算即可

**Yarn:** Yarn是Hadoop中的重要组件之一，负责海量数据运算时的资源调度

**Spark:** 大规模数据快速处理通用的计算引擎，提供大量的库Spark Core、Spark SQL、Spark Streaming、MLlib、GraphX等





## 10.1 Hadoop总体架构

**Flume:** Cloudera提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，用来做数据采集

**Kafka:** 分布式的消息发布/订阅系统，通过与Spark Streaming整合，完成实时业务计算

**Hive/Pig:** Hive是基于Hadoop的一个数据仓库工具，通过将结构化的数据文件（通常为HDFS文件）映射为一张数据表，提供简单的sql查询功能，将sql语句转换为MapReduce任务运行。Pig可以看做hadoop的客户端软件，可以连接到hadoop集群进行数据分析工作

**Hbase:** 建立在Hadoop文件系统之上的面向列的分布式数据库。不同于一般的关系数据库，它适合于存储非结构化的数据

**Redis:** 可基于内存也可以持久化的日志型、Key-Value数据库。往往用来缓存key-value类型的小表数据

**Oozie:** 一个可扩展的workflow系统，用于协调多个MapReduce作业的执行

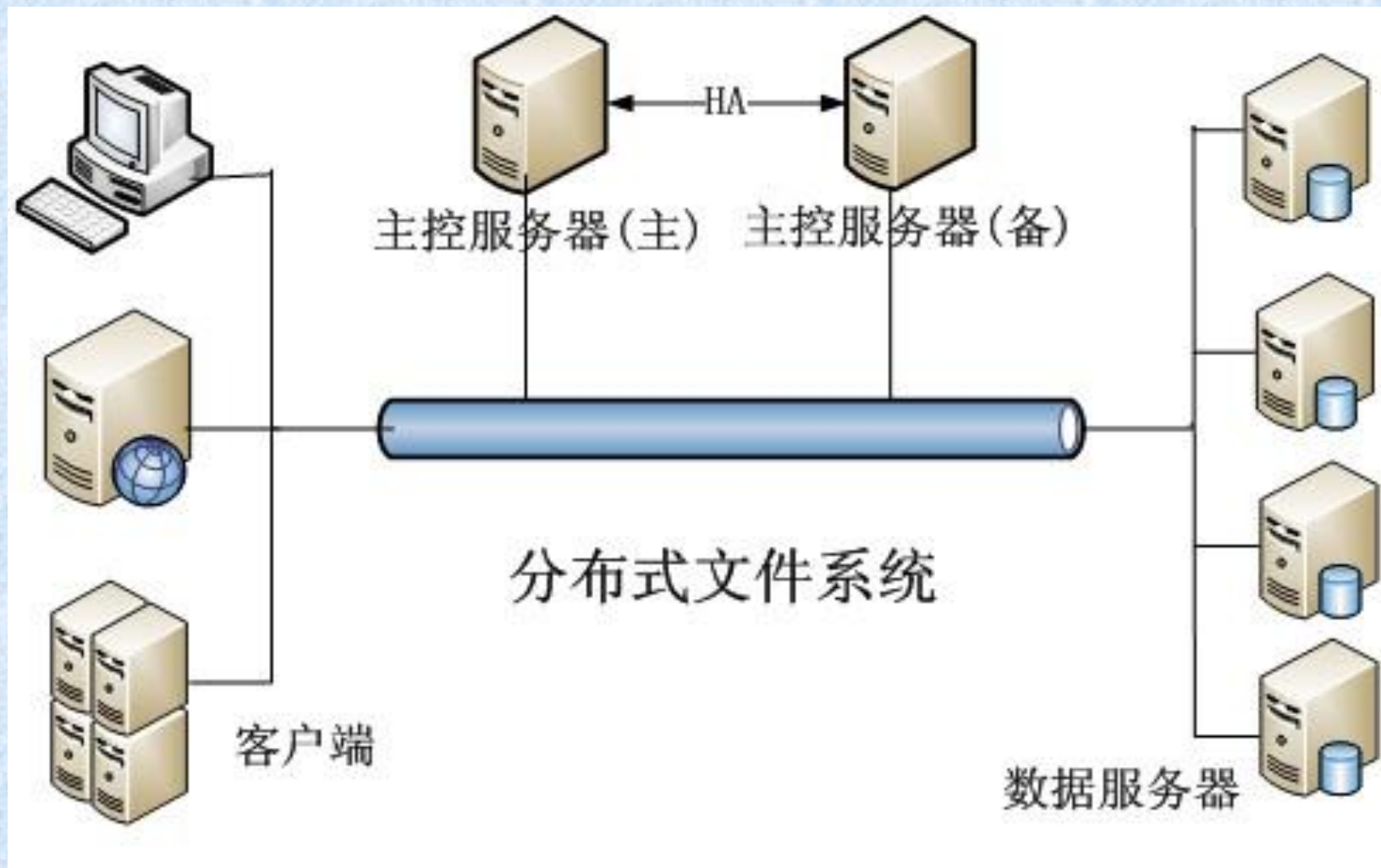




## 数据存储系统

- 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



## HDFS分布式文件系统

- 结构
  - 物理存储资源和对象分散在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

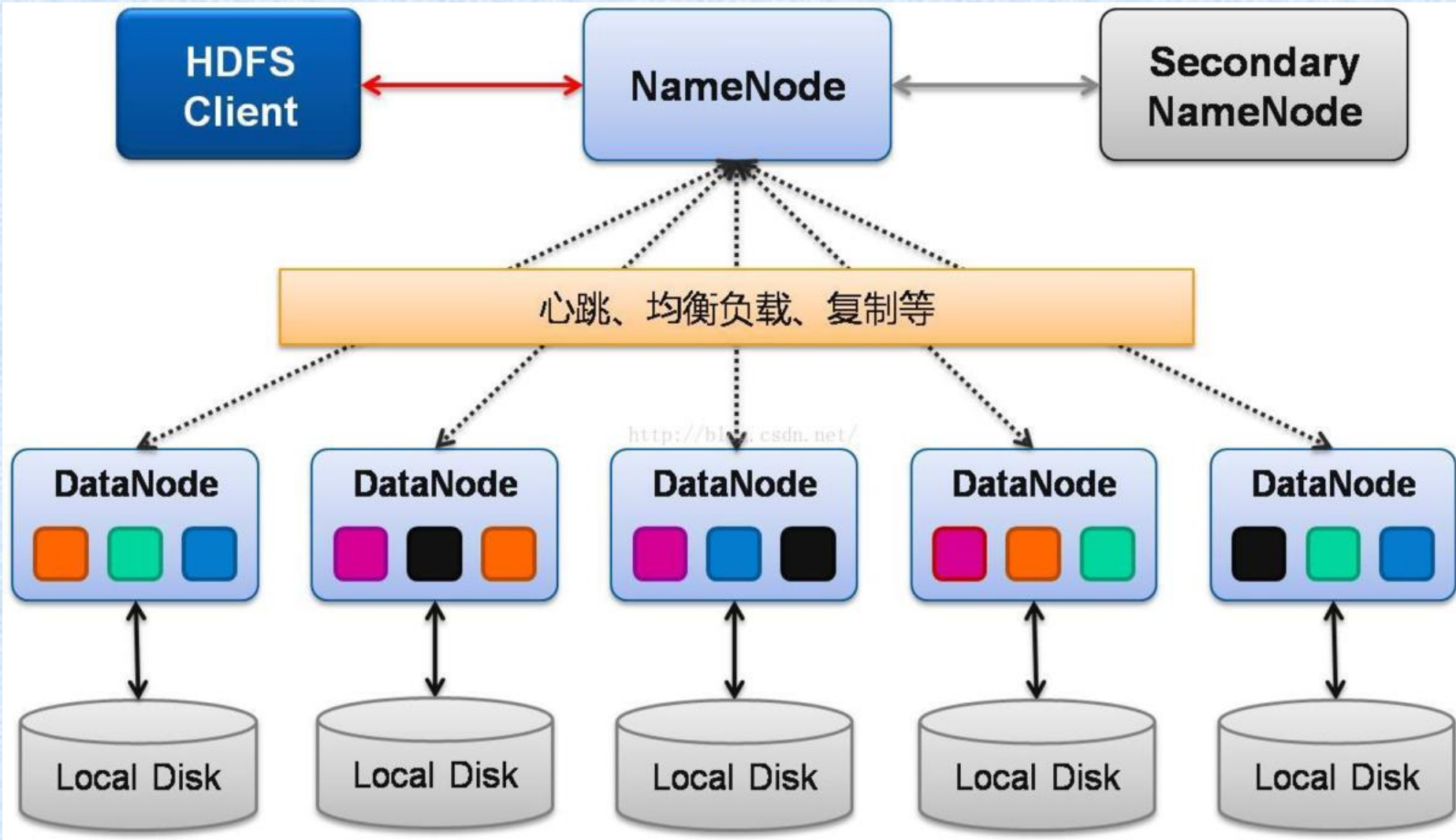




### HDFS体系结构

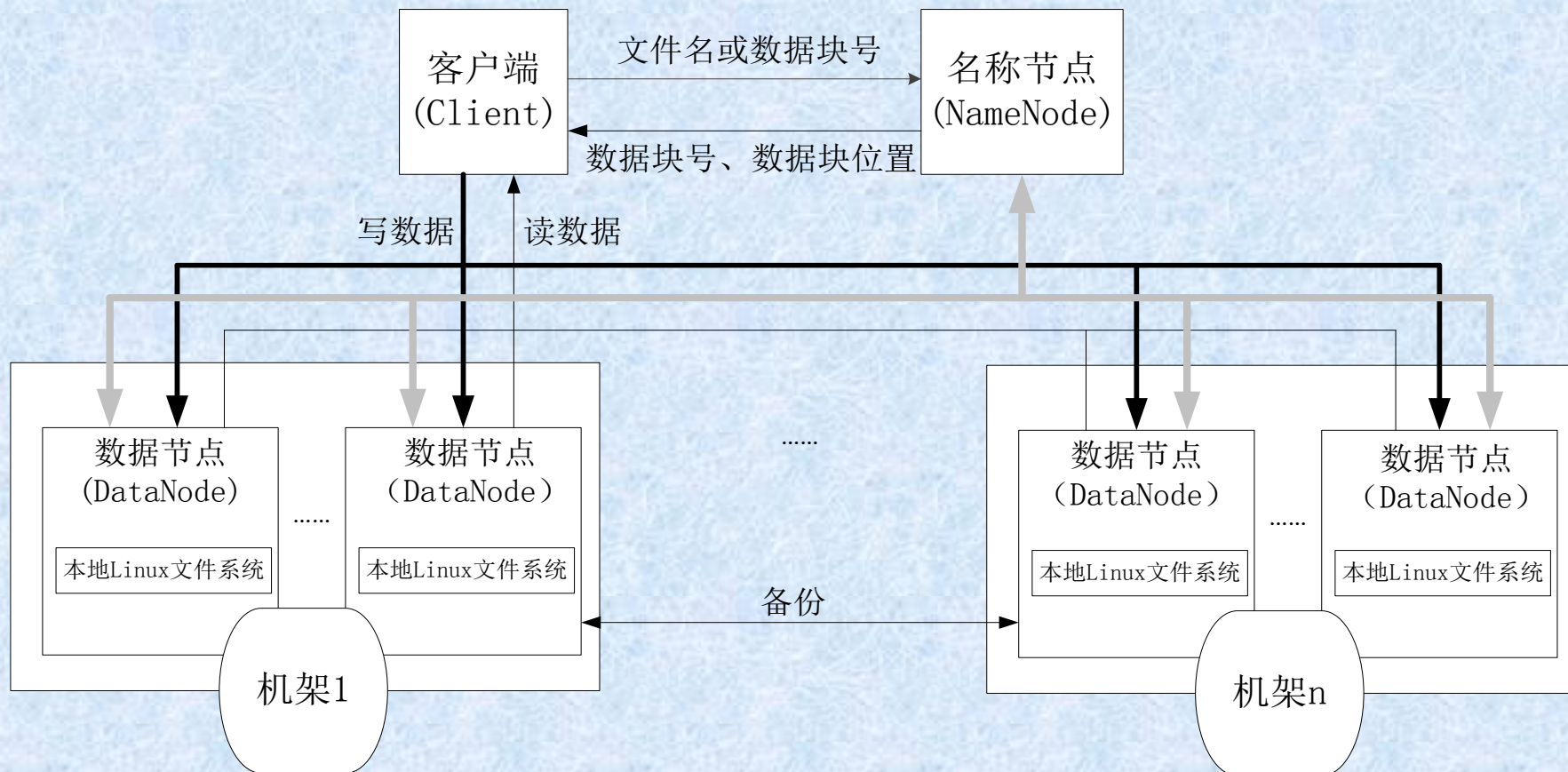
- 唯一主节点：运行NameNode，JobTracker，Zookeeper，Hmaster等负责集群管理、资源配置、作业调度的程序
- 多个从节点（dataNode）：承担数据存储及计算任务。
- 客户端（Client）：用于支持客户操作HDFS

## 10.2 HDFS文件系统





## 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



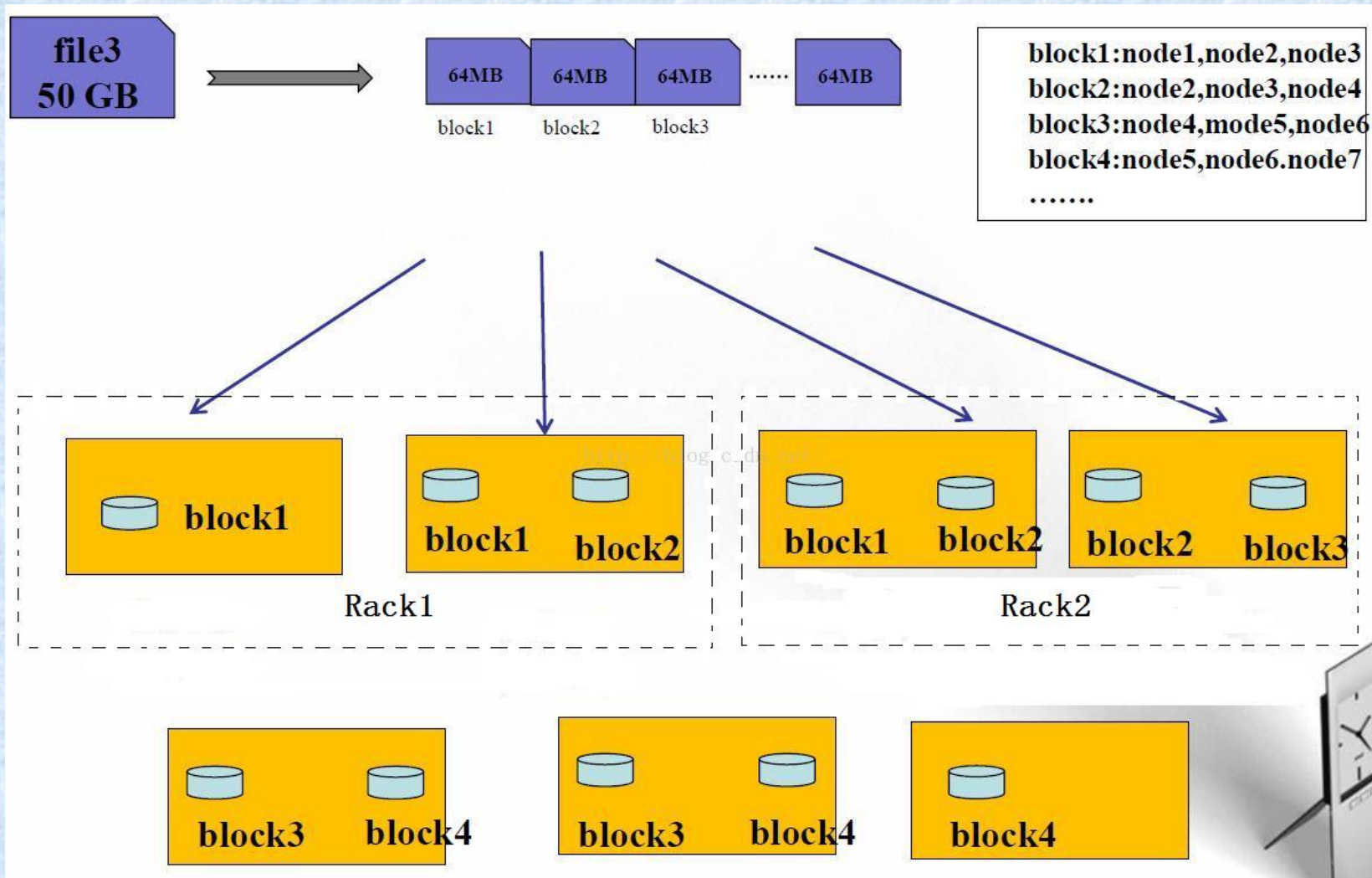


### HDFS存储结构

- 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



### 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





### 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



### 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



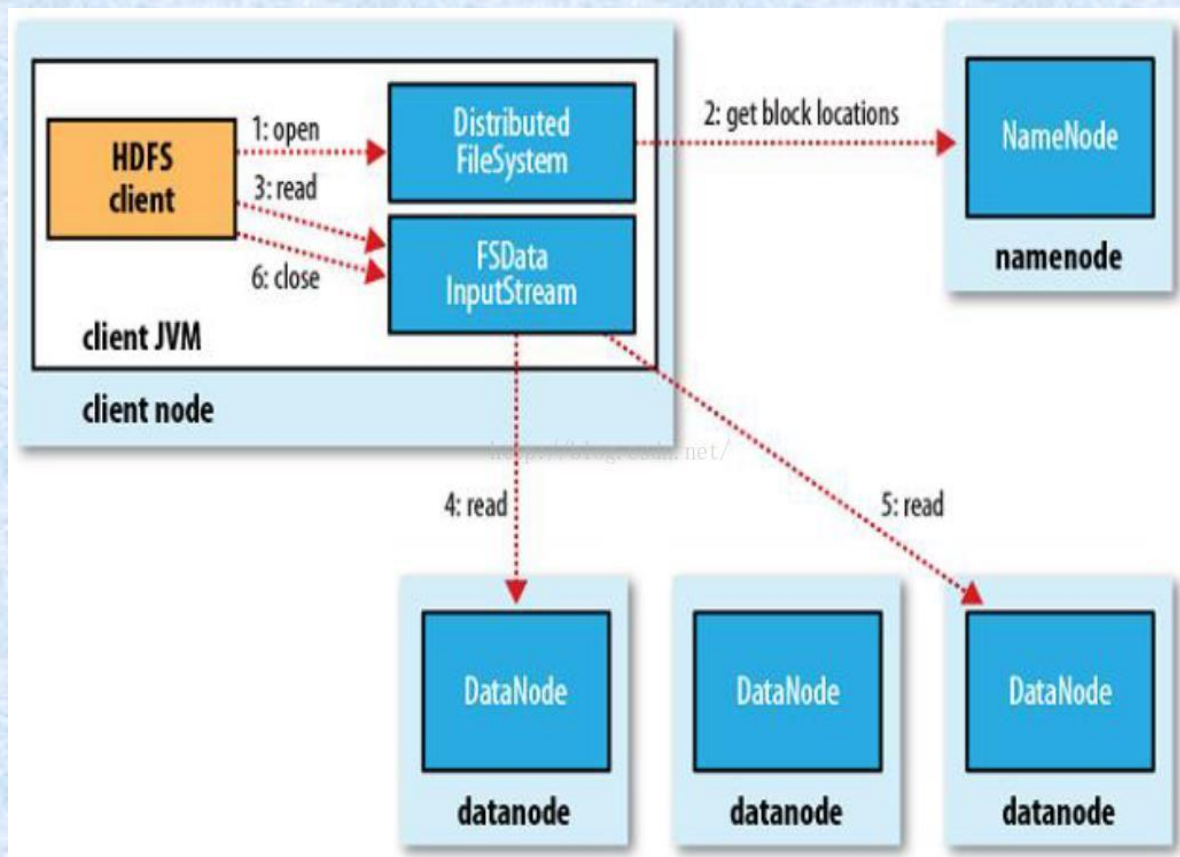
### HDFS文件读写机制

- 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

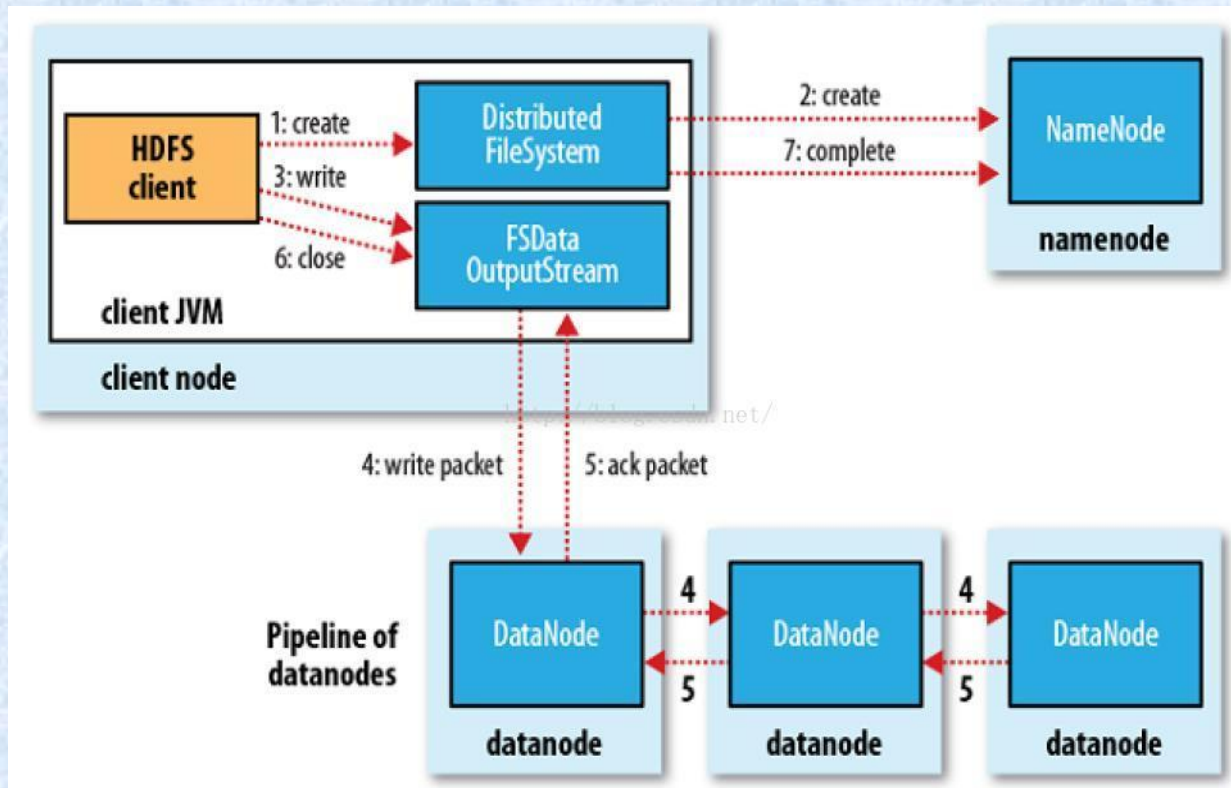




## 10.2 HDFS文件系统

### 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





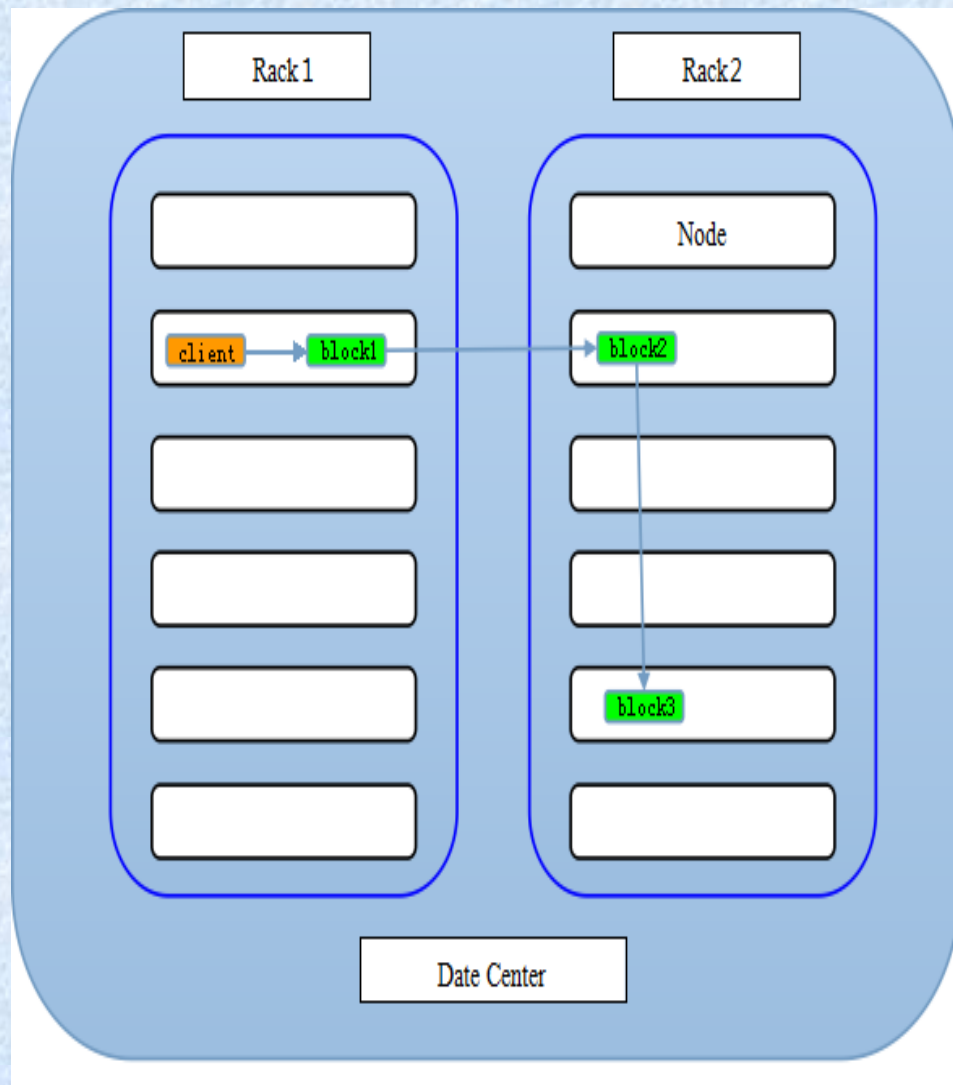
### HDFS数据容错与恢复

- 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

- 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





### 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





### 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



### 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



## EC纠删码技术(EC: Erasure Coding)

存储空间节省50%

HDFS是靠冗余存储提供容错性（数据块3倍存储），存储空间有效使用率只有33%。

EC编码技术可以达到两个目的：

- 1) 解决存储故障问题，提供容错性
- 2) 大幅度节省存储空间



## EC策略的原理

假设有三个值：

$$x_1 = 1$$

$$x_2 = 2$$

$$x_3 = 3$$

然后增加了三个方程

$$x_1 + x_2 + x_3 = 6$$

$$x_1 + 2x_2 + 4x_3 = 17$$

$$x_1 + 3x_2 + 9x_3 = 34$$

原理：上述6个方程，任意给我3个，我总可以解出全部x的值。





## EC技术的RS纠删码(Reed-Solomon Code)

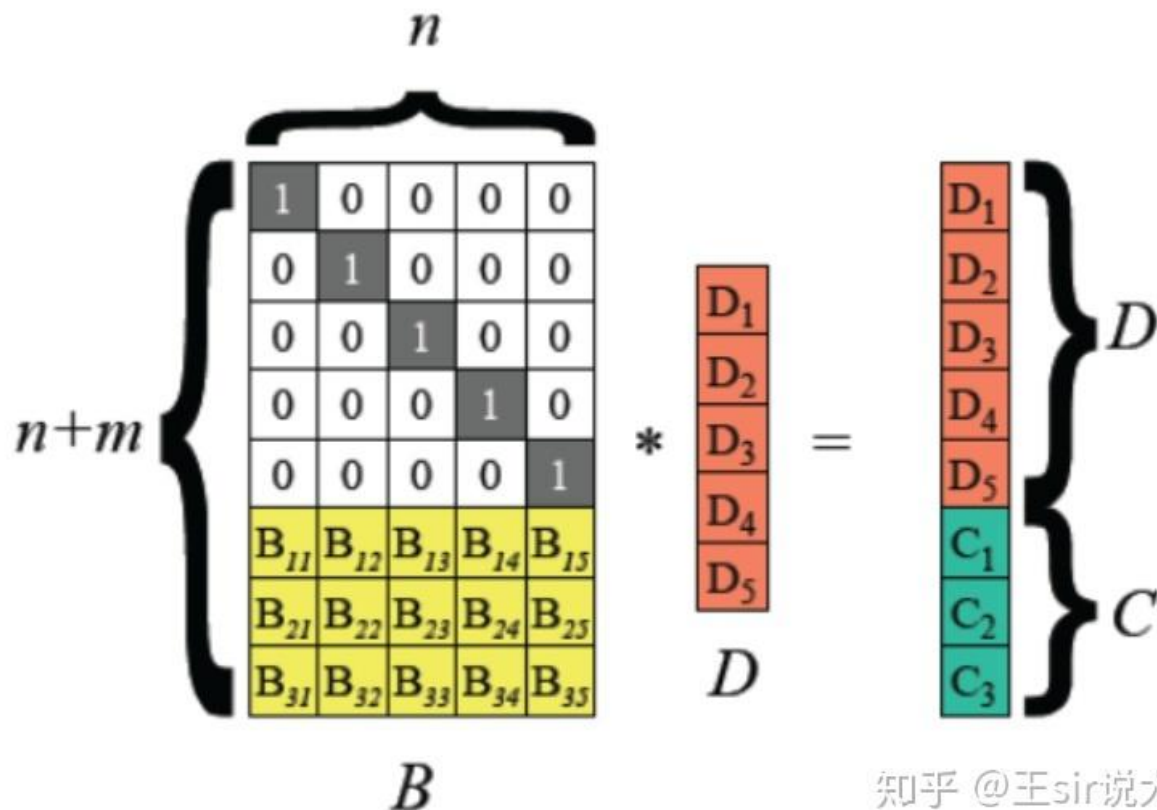
RS code是基于有限域的一种编码算法，有限域又称为 Galois Field，是以法国著名数学家伽罗华（Galois）命名的，在RS code中使用 $GF(2^w)$ ，其中 $2^w \geq n + m$ 。

**编码：**给定n个数据块（Data block） $D_1$ 、 $D_2$ …… $D_n$ ，和一个正整数m，RS根据n个数据块生成m个编码块（Code block）， $C_1$ 、 $C_2$ …… $C_m$ 。

**解码：**对于任意的n和m，从n个原始数据块和m个编码块中任取n块就能解码出原始数据，即RS最多容忍m个数据块或者编码块同时丢失。

# Hadoop 3.0 新特性

输入数据视为向量  $D = (D_1, D_2, \dots, D_n)$ ， $B$  为 RS 码，  
编码后数据视为向量  $(D_1, D_2, \dots, D_n, C_1, C_2, \dots, C_m)$ ，  
RS 编码可视为如图所示矩阵运算



$$\begin{matrix}
 & \overbrace{\hspace{1.5cm}}^n \\
 \underbrace{\hspace{1.5cm}}_{n+m} \begin{bmatrix}
 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 1 \\
 B_{11} & B_{12} & B_{13} & B_{14} & B_{15} \\
 B_{21} & B_{22} & B_{23} & B_{24} & B_{25} \\
 B_{31} & B_{32} & B_{33} & B_{34} & B_{35}
 \end{bmatrix}
 & * &
 \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{bmatrix}
 & = &
 \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}
 \end{matrix}$$

$B$

知乎 @王sir说大数据

## RS code编码数据恢复原理

RS最多能容忍 $m$ 个数据块被删除。数据恢复的过程如下：

1) 假设D1、D4、C2丢失，从编码矩阵中去除丢失的数据块/编码块对应的行

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1
$B_{11}$	$B_{12}$	$B_{13}$	$B_{14}$	$B_{15}$
$B_{21}$	$B_{22}$	$B_{23}$	$B_{24}$	$B_{25}$
$B_{31}$	$B_{32}$	$B_{33}$	$B_{34}$	$B_{35}$

$B$







上面的RS编码运算等式数学上变成如下的B' 以及等式

The diagram illustrates the mathematical representation of RS encoding. It shows a 5x5 matrix  $B'$  multiplied by a 5x1 vector  $D$  to produce a 5x1 vector of survivors.

The matrix  $B'$  is defined as:

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
$B_{11}$	$B_{12}$	$B_{13}$	$B_{14}$	$B_{15}$
$B_{31}$	$B_{32}$	$B_{33}$	$B_{34}$	$B_{35}$

The vector  $D$  is defined as:

$D_1$
$D_2$
$D_3$
$D_4$
$D_5$

The vector of survivors is defined as:

$D_2$
$D_3$
$D_5$
$C_1$
$C_3$

The equation is represented as:

$$B' * D = \text{Survivors}$$

知乎 @王sir说大数据





2) 由于 $B'$ 是可逆的, 记 $B'$ 的逆矩阵为 $B'^{-1}$ , 则 $B' * B'^{-1} = I$  单位矩阵。两边左乘 $B'$  逆矩阵

The diagram illustrates the multiplication of matrices  $B'$  and  $B'^{-1}$  to form the identity matrix  $I$ , followed by multiplication with matrix  $D$  to form matrix  $C$ .

Matrix  $B'^{-1}$  (5x5):

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0

Matrix  $B'$  (5x5):

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
$B_{11}$	$B_{12}$	$B_{13}$	$B_{14}$	$B_{15}$
$B_{31}$	$B_{32}$	$B_{33}$	$B_{34}$	$B_{35}$

Matrix  $D$  (5x1):

$D_1$
$D_2$
$D_3$
$D_4$
$D_5$

Matrix  $I$  (5x5):

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0

Matrix  $C$  (5x1):

$D_2$
$D_3$
$D_5$
$C_1$
$C_3$

The equation is:  $B'^{-1} * B' * D = I * D = C$

## 3) 得到如下原始数据D的计算公式

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

$I$

$D_1$
$D_2$
$D_3$
$D_4$
$D_5$

\*

=

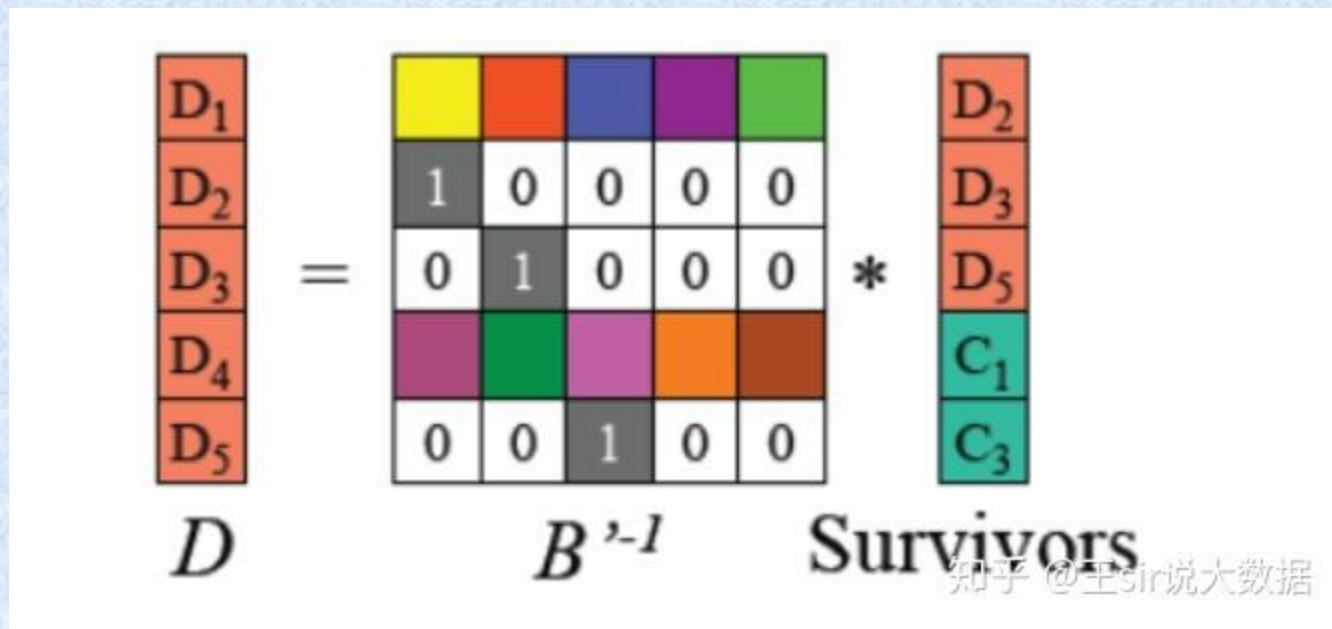
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0

\*

$D_2$
$D_3$
$D_5$
$C_1$
$C_3$

$B$  知乎 @王S 说大数据

也即原始数据D:



$$\begin{matrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ D_5 \end{matrix} = \begin{matrix} \text{Yellow} & \text{Red} & \text{Blue} & \text{Purple} & \text{Green} \\ \begin{matrix} 1 & 0 & 0 & 0 & 0 \end{matrix} \\ \begin{matrix} 0 & 1 & 0 & 0 & 0 \end{matrix} \\ \text{Purple} & \text{Green} & \text{Purple} & \text{Orange} & \text{Brown} \\ \begin{matrix} 0 & 0 & 1 & 0 & 0 \end{matrix} \end{matrix} * \begin{matrix} D_2 \\ D_3 \\ D_5 \\ C_1 \\ C_3 \end{matrix}$$

$D \qquad B^{n-1} \qquad \text{Survivors}$

知乎 @王sir说大数据

4) 依据上述公式对有缺失的编码重新计算，可以恢复得到丢失的数据