Hadoop平台

BIG DATA

Hadoop平台

CONTENTS

01 Hadoop概述
Overview of Hadoop

02 HDFS简介
Introduction of MapReduce

03 YARN简介
Introduction of HDFS

04 MapReduce简介
Overview of Spark

1.1 Hadoop简介

- 开源
  - Hadoop是Apache软件基金会旗下的一个开源分布式计算平台，为用户提供了系统底层细节透明的分布式基础架构，在分布式环境下提供了海量数据的处理能力。
- JAVA
  - Hadoop基于Java开发的，具有优异跨平台特性，并可部署在廉价的计算机集群中
- 核心：HDFS+MapReduce
  - HDFS（Hadoop Distributed File System）：分布式文件系统，用于存储海量数据。
  - MapReduce：并行计算框架，用于海量数据上的高效计算。

几乎所有主流厂商围绕Hadoop提供开发工具、开源软件、商业化工具和服务

1.1.1 Hadoop的产生背景

- 数据爆炸式增长，数据处理速度却没有跟上
  - 纽约证交所每天产生的交易数据约在4TB至5TB之间；
  - Facebook存储的照片超过2400亿张，且每月以至少7PB的速度增长；
  - 人类科学家合成首张黑洞照片所用的数据为5PB
- 硬盘存储容量不断提升，但访问（读/写）速度却没有与时俱进
  - 1990年，一个普通硬盘容量为1370MB，传输速度为4.4MB/s，读完一个硬盘只需要5分钟。
  - 如今，4T硬盘成为主流，传输速度约为100MB/s，读完整个硬盘需要10个小时。
- 分布式存储和并行计算所带来的新挑战
  - 硬件故障问题，如何在硬件故障发生时避免数据丢失。
  - 协同问题，如何在不同的计算节点之间协同，保证计算结果的准确性。

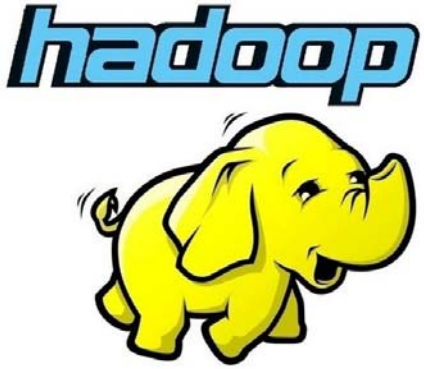
Hadoop——可靠且可扩展的存储和分析平台！

1.1.2 Hadoop名字的来源

- Hadoop不是缩写，这个词是生造出来的。Hadoop之父Doug Cutting这样解释Hadoop的来历：

> “这个名字是我的孩子给他的毛绒玩具取得。我的命名标准是好拼读，含义宽泛，不会被用于其他地方。小朋友是这方面的高手。”

- Doug Cutting——Hadoop创始人
  - 1985 年毕业于美国斯坦福大学。大学时代一开始学习物理、地理等课程，后迫于学费压力，转学计算机软件。
  - 1997年开发了Lucene——第一个开源全文文本检索函数库；
  - 2002年与Mike Cafarella启动Nutch项目，用于大规模网页爬取和索引；
  - 2004年，在Nutch上实现了NDFS和MapReduce；
  - 2006年，将NDFS和MapReduce从Nutch独立出来，命名为Hadoop。



1.1.3 Hadoop发展简史

Doug Cutting在Nutch开始
研究DFS和MapReduce

Doug Cutting & Mike
Cafarella创建Nutch

2002 2003 2004 2005 2006 2007 2008 2009 2011

Google发布GFS&
MapReduce论文

Cutting跳槽到Yahoo!
并带走了Hadoop

纽约时报在100个
Ec2s上转换了4TB的图片

使用910节点对1TB
数据排序，耗时3.5mins

1TB,62secs,1460
1PB,16.25hours,3658

HortonWorks成立

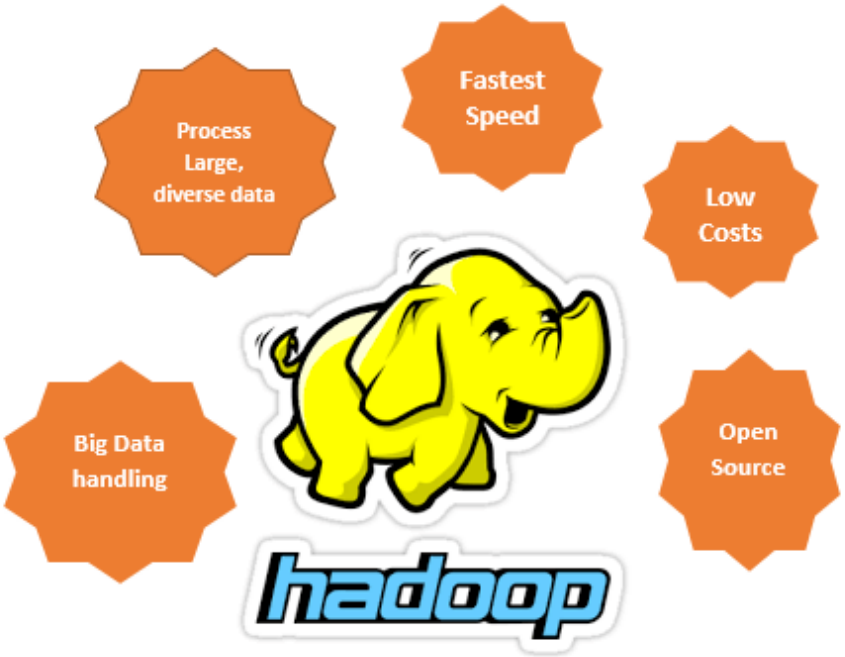
Cloudera成立

Facebook创建构建在
Hadoop之上的SQL: Hive

Cutting加入
Cloudera

1.2 Hadoop特性

Hadoop是一个能够对大量数据进行分布式处理的软件框架，并且是以一种可靠、高效、可伸缩的方式进行处理的，它具有高效性，成本低，开源等特性。



1.2 Hadoop特性

问题思考：为什么不用配有大量硬盘的传统关系型数据库来进行海量数据分析？

- 计算机硬盘特点：寻址时间的提升远远赶不上传输速度的提升。
  - 寻址：即将磁头移动到特定的磁盘位置进行读写操作，是导致硬盘操作延迟的重要原因。
  - 传输速率主要取决于硬盘带宽。
- 传统关系型数据库
  - 访问模式中包含大量硬盘寻址，数据读取需要耗费更多时间。
  - 基于B树结构，当数据库系统中有大量数据更新时，B树效率明显下降。
- Hadoop的MapReduce计算框架
  - 比较适合解决需要以批处理方式分析整个数据集的问题，如一些特定的分析任务。
  - 一次写入，多次读取。

1.2 Hadoop与传统关系型数据库的区别

- 传统关系型数据库主要针对结构化数据，Hadoop更擅长处理非结构化数据。

	传统的关系型数据库	MapReduce
数据大小	GB	PB
数据存取	交互式和批处理	批处理
更新	多次读/写	一次写入，多次读取
事务	ACID	无
结构	写时模式	读时模式
完整性	高	低
横向扩展	非线性的	线性的

1.3 MapReduce基本思想

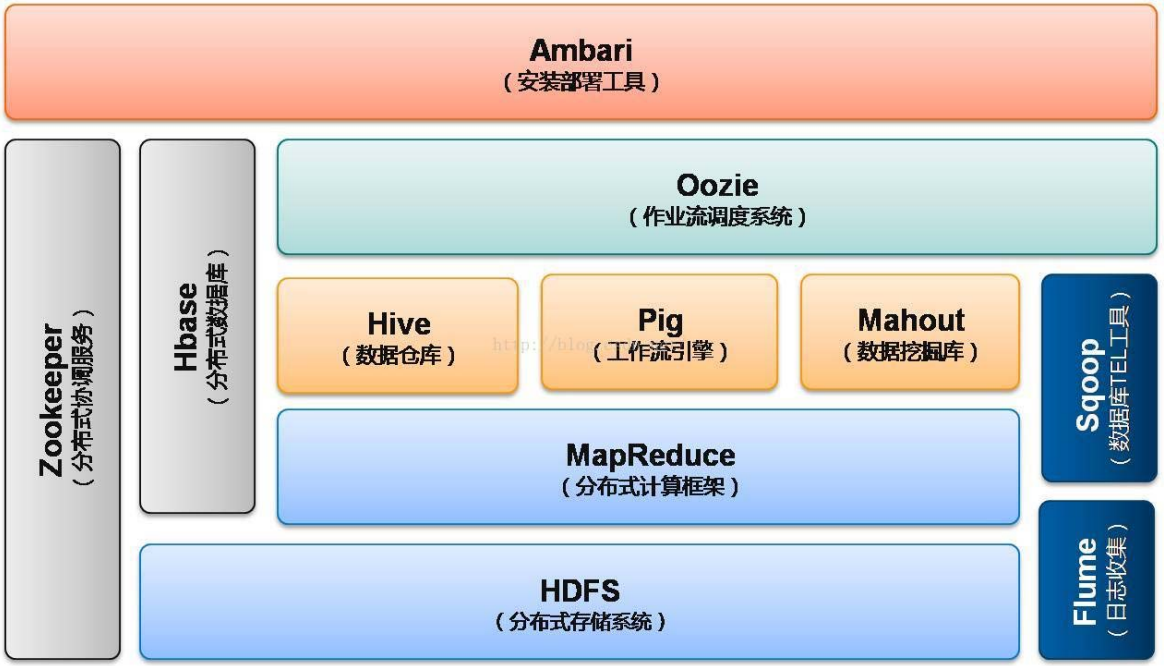
- MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。
  - Map函数：主要输入是一对<key，value>值，经过map计算后输出一对<key，value>值；
  - MapReduce框架将相同key合并，形成<key，value集合>；
  - Reduce函数：以<key，value集合>作为输入，经过计算输出零个或多个<key，value>对。
- 例子：词频统计

1.4 Hadoop应用现状

Hadoop凭借其突出的优势，已经在各个领域得到了广泛的应用，而互联网领域是其应用的主阵地。

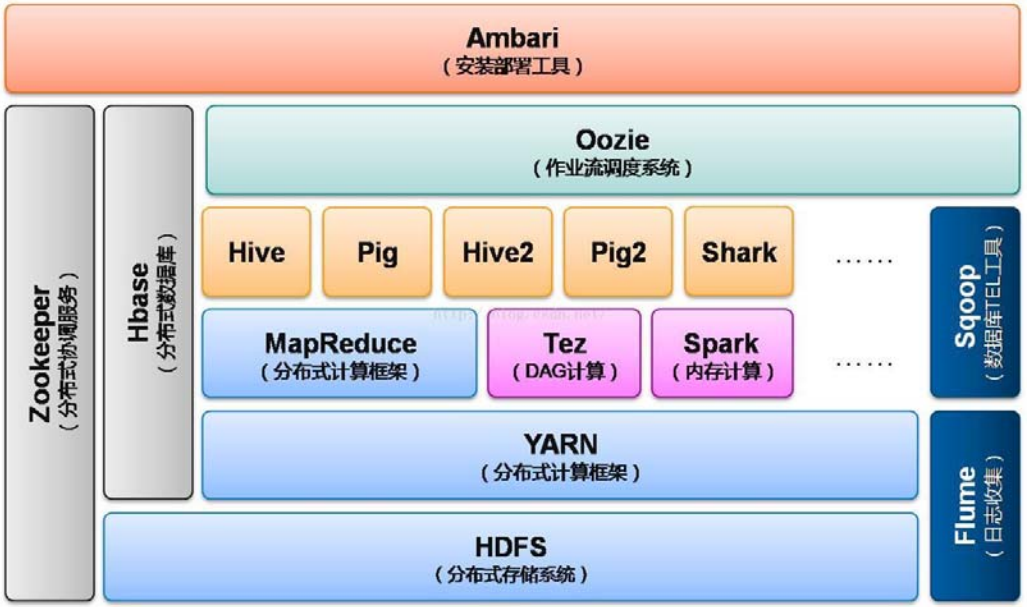
1.5 Hadoop架构（1.X）

- hadoop1.0时期架构



1.5 Hadoop架构（2.X）

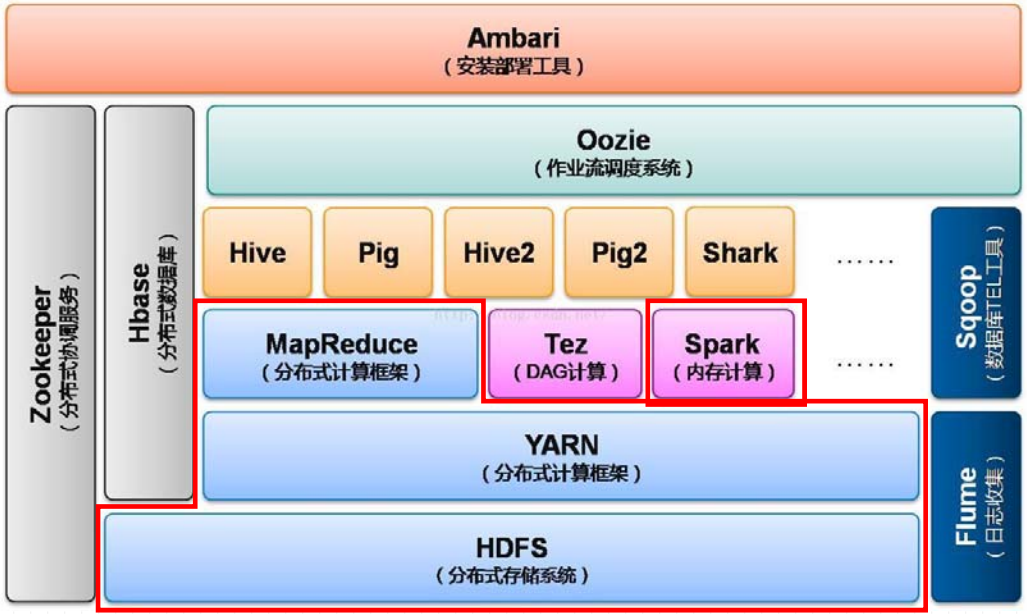
- hadoop2.0时期架构，引入YARN集群资源管理框架



组件	功能
HDFS	分布式文件系统
MapReduce	分布式并行编程模型
YARN	资源管理和调度器
Tez	DAG（有向无环图）计算框架
Hive	Hadoop上的数据仓库
HBase	Hadoop上的非关系型的分布式数据库
Pig	一个基于Hadoop的大规模数据分析平台，提供类似SQL的查询语言Pig Latin
Sqoop	用于在Hadoop与传统数据库之间进行数据传递
Oozie	Hadoop上的工作流管理系统

1.5 Hadoop架构（2.X）

- hadoop2.0时期架构，引入YARN集群资源管理框架



组件	功能
Zookeeper	提供分布式协调一致性服务
Storm	流计算框架
Flume	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
Ambari	Hadoop快速部署工具，支持Apache Hadoop集群的供应、管理和监控
Kafka	一种高吞吐量的分布式发布订阅消息系统
Spark	类似于Hadoop MapReduce的通用并行框架

1.6 Hadoop发行版本

- 由于Hadoop本身是一个开源框架，因此不同厂商对其进行完善，提供了不同的发行版本：
  - **Apache Hadoop**：最原始的版本，所有发行版均基于这个版本进行改进
  - **Cloudera**版本：Cloudera’s Distribution Including Apache Hadoop，简称CDH
  - **Hortonworks**版本：Hortonworks Data Platform，简称“HDP”

\	Apache Hadoop	CDH	HDP
开源情况	100%开源	100%开源	100%开源
收费情况	完全免费	免费版和企业版	完全免费
管理工具	Apache Ambari	Cloudera Manager	Ambari
稳定性	中	高	高
运维成本	高	中	中
生态支持	兼容性差	完善	完善

Hadoop平台

CONTENTS

01 Hadoop概述
Overview of Hadoop

02 HDFS简介
Introduction of MapReduce

03 YARN简介
Introduction of HDFS

04 MapReduce简介
Overview of Spark

3.1 HDFS分布式文件系统

- **HDFS（Hadoop Distributed File System）**
- HDFS是Hadoop下的分布式文件系统，具有高度容错性，适合部署在廉价的机器上。
- HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。
- HDFS把文件分布存储到多个计算机节点上，成千上万的计算机节点构成计算机集群。
- 与以往使用多个处理器和专用高级硬件的并行化处理装置不同的是，目前的分布式文件系统所采用的计算机集群，都是由普通商用硬件构成的，这就大大降低了硬件上的开销。

HDFS的设计特点

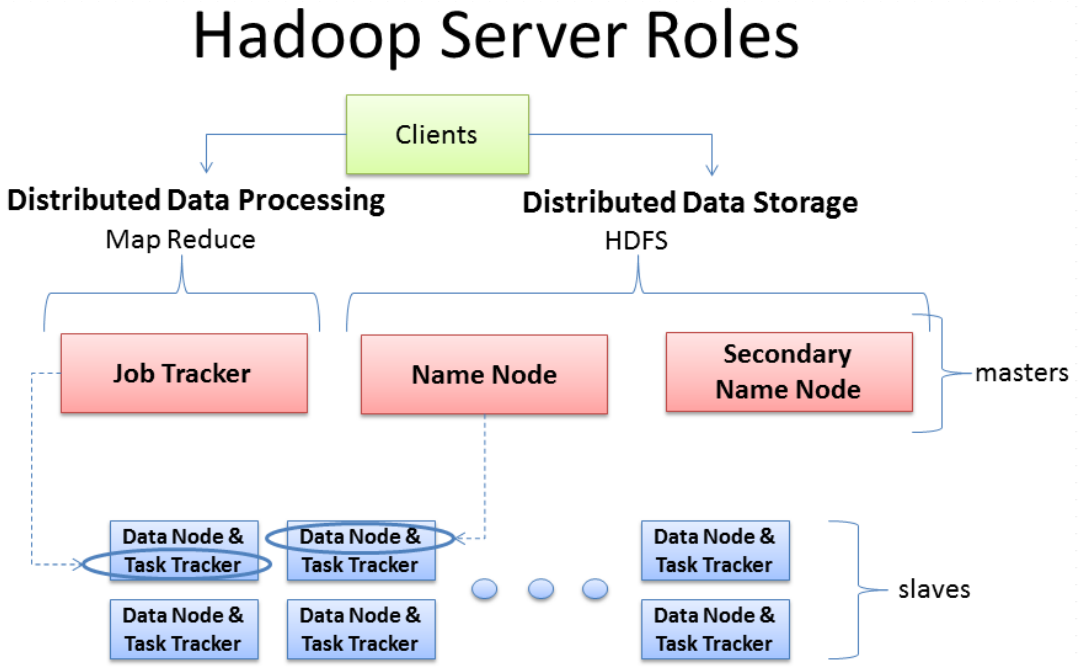
- **超大文件**：这里“超大文件”指具有几百MB、几百GB甚至几百TB大小的文件。
- **流式数据访问**：一次写入、多次读取是最高效的访问模式。数据集通常由数据源生成或直接复制而来，每次分析都涉及数据集的大部分甚至全部。
- **商用硬件**：不需要运行在昂贵且高可靠的硬件上，允许节点出现故障，节点故障时系统能够继续运行且不让用户觉察到明显的中断。

以下应用场景不适合HDFS

- **低时间延迟的数据访问**：HDFS主要为高数据吞吐量应用进行优化，可能会以提高时间延迟为代价，不适合有低时延要求的应用。
- **大量的小文件**：HDFS需要在namenode的内存中维护所有文件的元数据，大量小文件容易导致其内存耗尽。
- **多用户写入，任意修改文件**：HDFS中的文件不支持多个写入者操作，也不支持在文件任意位置进行修改。

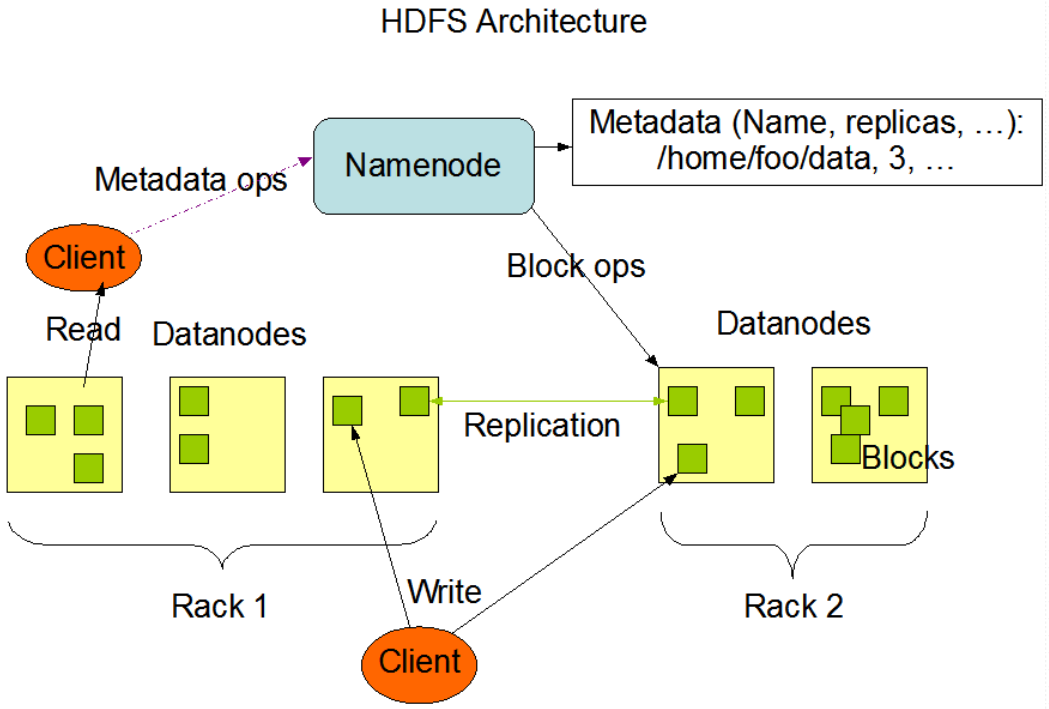
2.1 分布式文件系统

- HDFS由计算机集群中多个节点构成，一类叫“主节点”(Master Node)或称“名称结点”(NameNode)，另一类叫“从节点”（Slave Node）或称“数据节点”(DataNode)



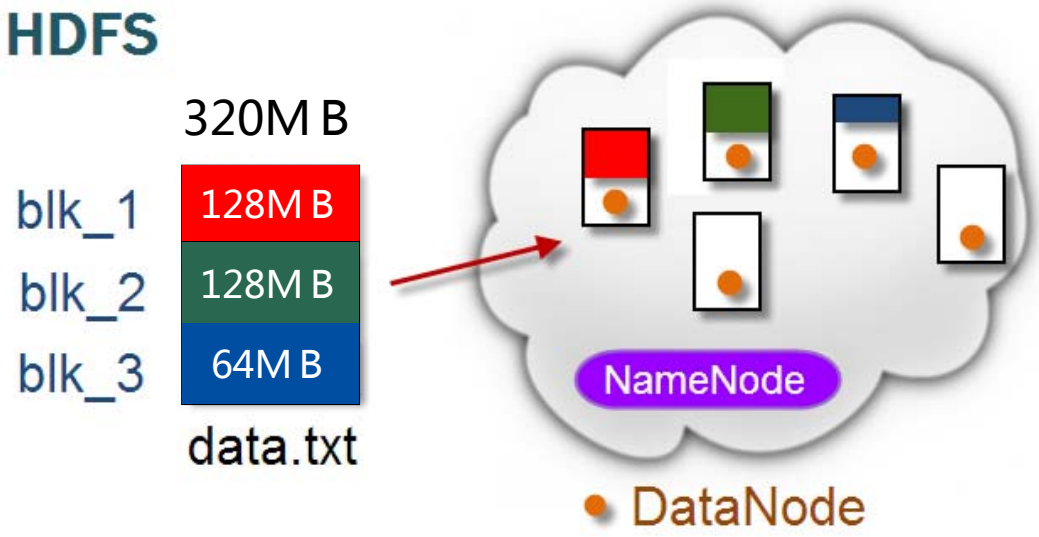
2.2.1 HDFS体系结构

- Master/Slave模型
  - 一个HDFS集群含一个名称节点（NameNode）和若干数据节点（DataNode）
  - NameNode作为中心服务器，负责管理文件系统命名空间及客户端文件访问
  - DataNode一般一个节点运行一个进程，处理文件系统客户端读/写请求，在NameNode统一调度下进行数据块创建、删除、复制等。每个DataNode的数据实际上是保存在本地Linux文件系统中



2.2.2 HDFS块（Chunk）

- HDFS把一个大文件分为多个数据块，以块作为存储单位
- 默认一个块大小为**128MB**（Hadoop 1.x版本是64MB）



思考：HDFS数据块为什么这么大？

文件分块的好处

- **支持大规模文件存储。**文件以块为单位存储，一个大文件可分拆若干文件块，不同文件块可分发不同节点上。文件大小不会受到单个节点存储容量的限制，可远大于网络任意节点容量
- **简化系统设计。**因为文件块大小固定，易计算出一个节点可存储多少文件块；其次，方便元数据管理，元数据不要和文件块存储，可以由其他系统负责管理元数据
- **适合数据备份。**每个文件块都可冗余存储到多节点上，大大提高了系统容错性和可用性

HDFS块（Chunk）为什么这么大？

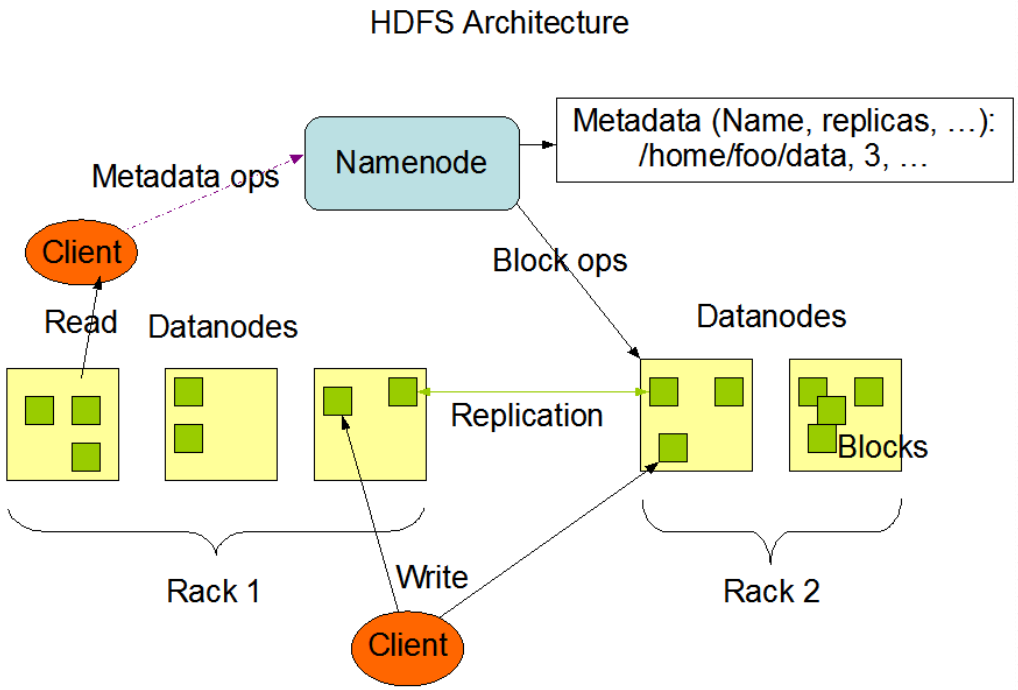
- **原因1：最小化寻址开销**
  - 如果块足够大，从磁盘传输数据的时间会明显大于寻址（定位这个块开始位置）所需时间，因此传输大文件的时间将主要取决于磁盘传输速率，不受寻址开销影响。
  - 例子：假设平均寻址时间为10ms，磁盘传输速率为100MB/s，为了使寻址时间仅占传输时间的1%，需要将块大小设置为多少？
  - 答案：100MB

思考：HDFS数据块是不是越大越好？

- **原因2：减少NameNode节点内存开销**
  - NameNode节点需要在内存中维护所有文件的元数据信息，因此HDFS能够支持的文件个数取决于NameNode内存大小。
  - 例子：通常每个文件、目录、数据块的元数据占150字节，若有100万个文件，且每个文件占一个数据块，则至少需要300MB内存。

2.2.3 HDFS NameNode和DataNode

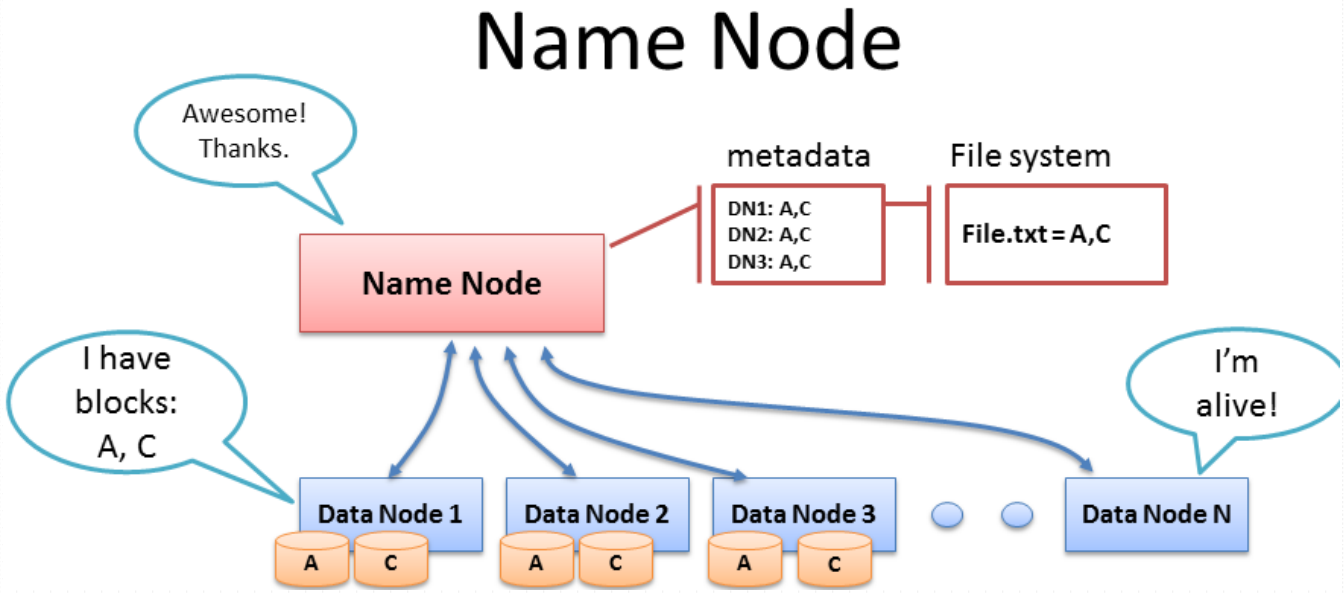
- **NameNode**
  - 存储元数据，元数据保存在内存中
  - 保存文件、block、DataNode之间的映射关系
- **DataNode**
  - 存储文件内容，文件内存保存在磁盘中
  - 维护了block id到DataNode之间的映射关系
- **客户端Client**
  - 代表用户通过与NameNode和DataNode交互来访问整个文件系统，屏蔽底层细节



2.2.3 HDFS NameNode和DataNode

NameNode

NameNode掌管集群所有文件系统元数据（metadata），并监督DataNode存活状态及协调数据的存取。 NameNode是HDFS的中央控制中心，自身不储存数据，只知道某个文件是由那些block组成，而那些block又存放在cluster的那个位置(DataNode)。



NameNode会维护文件系统树及整棵树内所有的文件和目录，这些信息以两个文件的形式存储到磁盘上：命名空间镜像文件和编辑日志文件

NameNode并不会永久保存块的位置信息，因为这些信息会在系统重启时根据DataNode的信息重建。

DataNode定期向NameNode发送：

- 自己存储的块列表
- 心跳信息：自己还活着，剩余存储

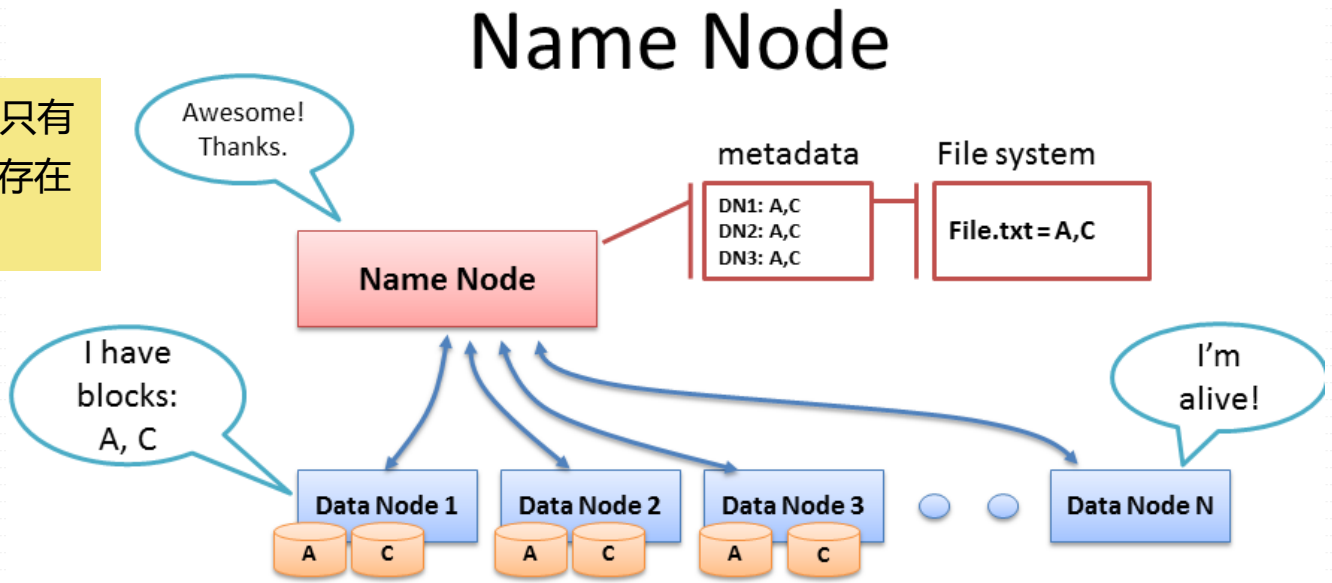
2.2.3 HDFS NameNode和DataNode

NameNode

Namenode指引客户端（Client）需要跟哪些Datanode联系，记录集群储存容量，每个Datanode存活状态。Datanode每3秒由TCP handshark向Namenode传送heartbeat信息，每10次的heartbeat传送block report，告知Namenode全部储存的block有哪些。

思考：这个体系架构中只有一个Name Node，会存在什么问题？

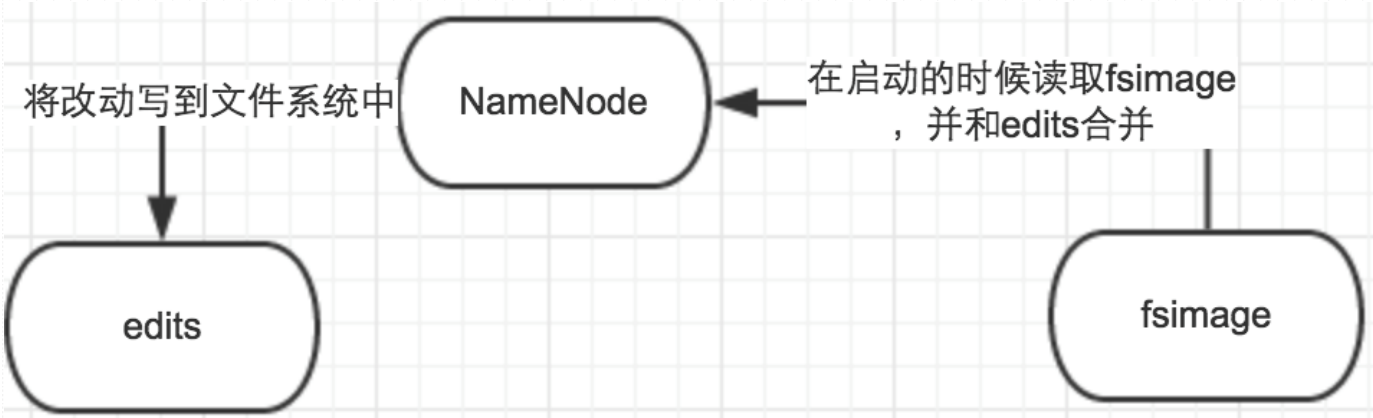
单点故障



2.2.3 HDFS NameNode和DataNode

NameNode中的FSImage和Edits文件

- FSImage：记录某一永久性检查点（Checkpoint）时整个HDFS的元信息（系统快照）。启动HDFS时，NameNode会读取FSImage并与Edits合并，构建HDFS目录结构并加载到NameNode内存中
- **Edits**：HDFS系统启动之后，对HDFS的操作步骤都会记录到编辑日志Edits文件中
- Checkpoint机制：HDFS会定期（默认1小时）通过Secondary Namenode将最近的fsimage和edits文件合并到新的fsimage中，Namenode启动时会把最新的fsimage加载到内存中。



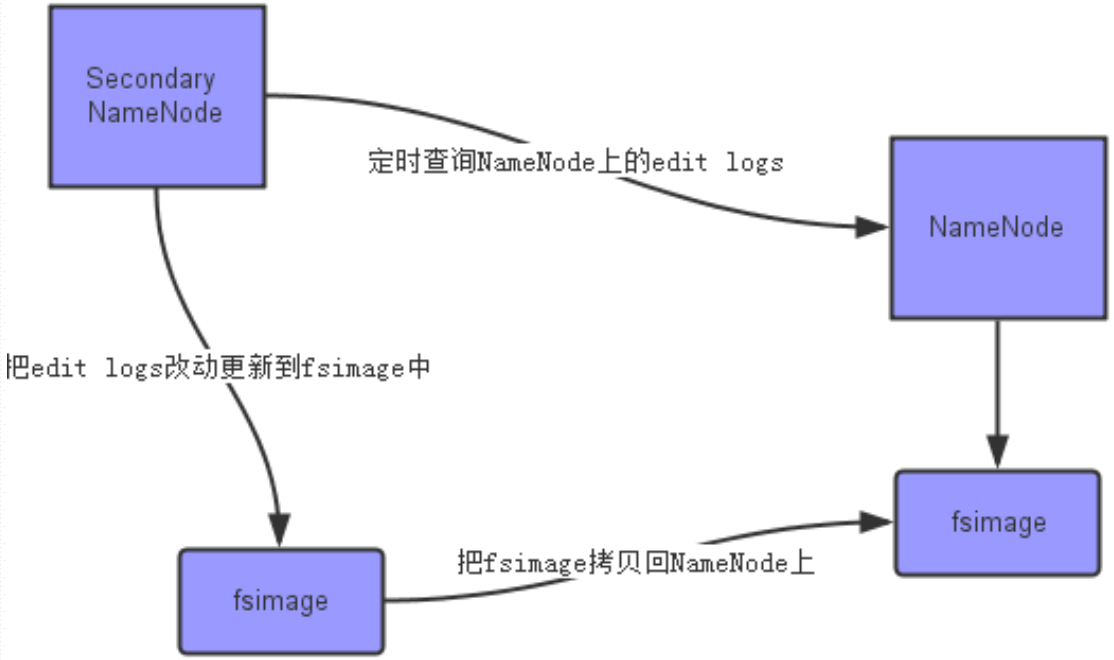

```
edits_0000000000000008801-0000000000000008802
edits_0000000000000008803-0000000000000008803
edits_0000000000000008804-0000000000000008804
edits_0000000000000008805-0000000000000008833
edits_0000000000000008834-0000000000000008835
edits_0000000000000008836-0000000000000008837
edits_0000000000000008838-0000000000000008839
edits_0000000000000008840-0000000000000008841
edits_0000000000000008842-0000000000000008843
edits_0000000000000008844-0000000000000008845
edits_0000000000000008846-0000000000000008847
edits_0000000000000008848-0000000000000008852
edits_0000000000000008853-0000000000000008923
edits_0000000000000008924-0000000000000008925
edits_0000000000000008926-0000000000000008927
edits_inprogress_0000000000000008928
fsimage_0000000000000008925
fsimage_0000000000000008925.md5
fsimage_0000000000000008927
fsimage_0000000000000008927.md5
```

2.2.3 HDFS NameNode和DataNode

Secondary NameNode

为什么需要Secondary Namenode？

- 在生产环境中NameNode很少重启，因此当NameNode运行了很长时间后，Edits文件会变得很大，下次启动时需要很长时间重构文件系统结构
- 如果NameNode挂掉，那就丢会失很多改动因为此时的fsimage文件非常旧
- Secondary NameNode的职责：定时到NameNode去获取edit logs，将edit logs更新到自己的fsimage上，并将最新的fsimage拷贝会NameNode



思考：为什么不由NameNode自己完成这个工作？

2.2.3 HDFS NameNode和DataNode

- **NameNode备份(HA)机制**

NameNode是HDFS的核心，一旦NameNode宕机，整个系统将不可用

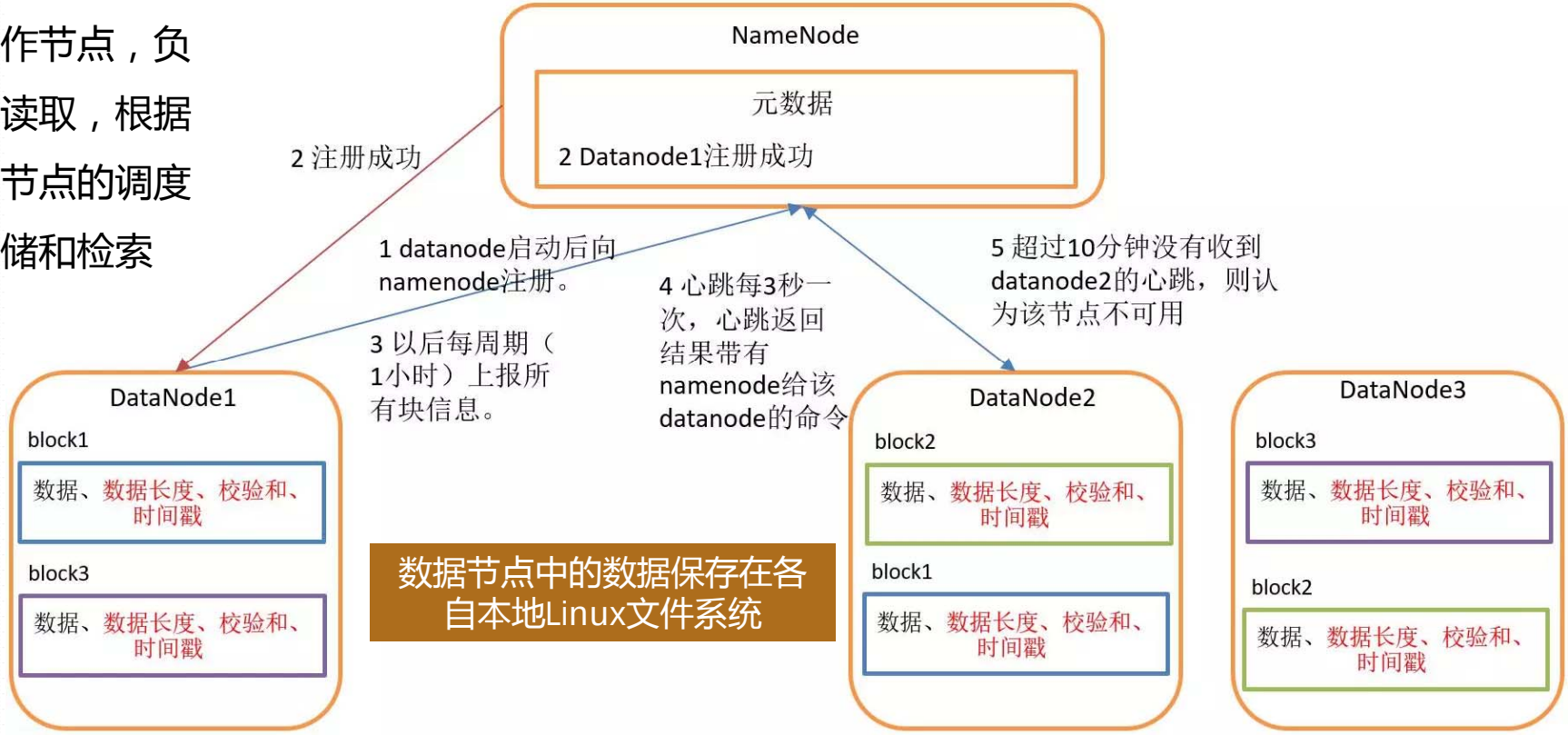
- **基本原理：**
  - 两个namenode，一个active namenode，状态为active，另一个standby namenode，状态是standby
  - 两者的状态可以切换，但只有1个为active状态，只有active namenode提供对外的服务，当active namenode失效时，standby namenode接管服务
  - active namenode和standby namenode之间通过NFS或者QJM方式来同步数据
- **NFS（Network File System）方式**
  - active namenode会把最近的edits文件写到NFS，standby namenode从NFS中把数据读过来。
  - 缺点：若active或者standby和NFS之间网络有问题，则会造成他们之前数据的同步出问题
- **QJM（Quorum Journal Manager ）方式**

自行查阅资料了解

2.2.3 HDFS NameNode和DataNode

DataNode

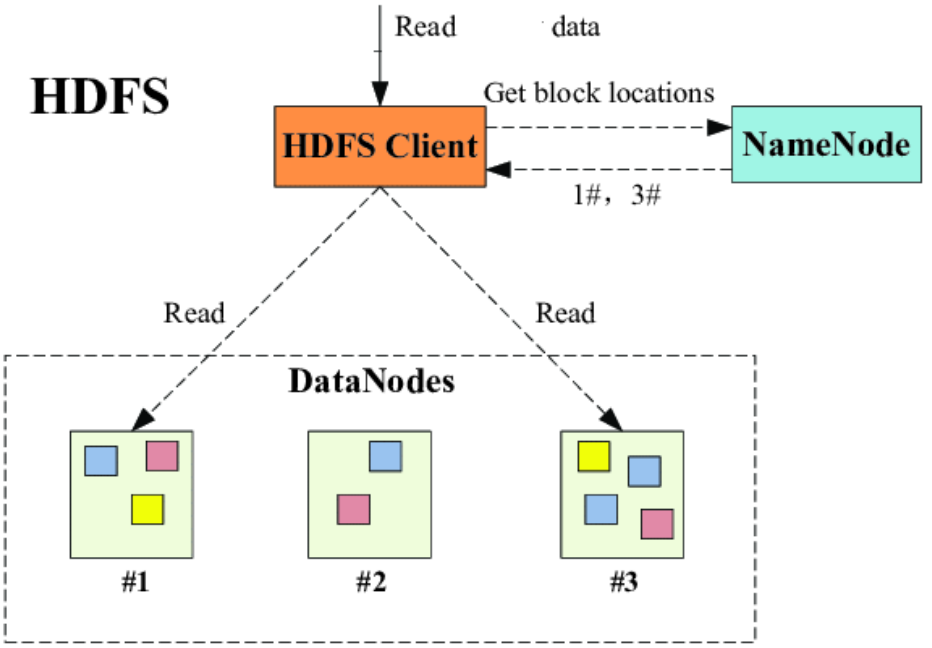
- 数据节点是工作节点，负责数据存储和读取，根据客户端或名称节点的调度来进行数据存储和检索



2.3 HDFS存储原理

a) 冗余数据保存

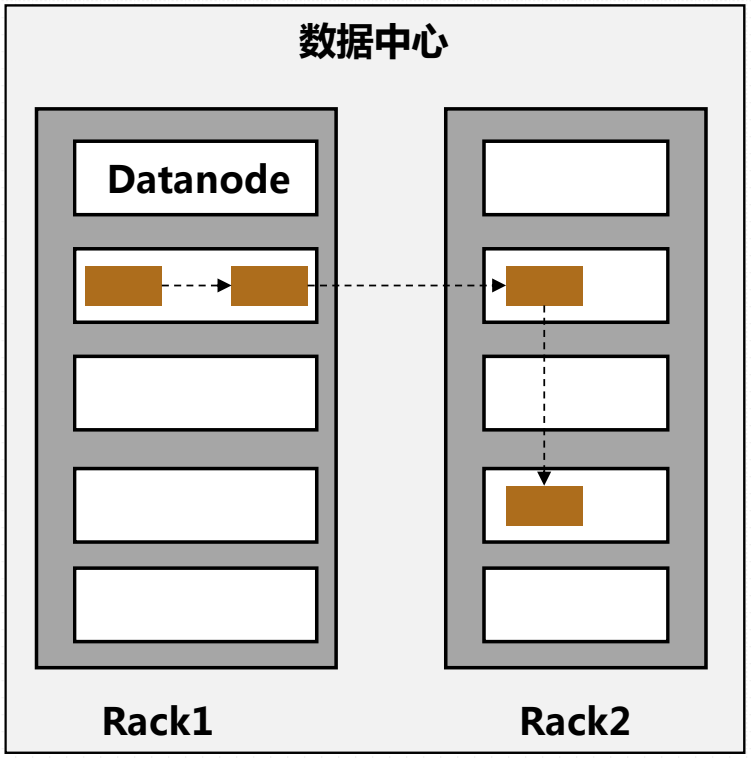
- 为了保证系统容错性和可用性，HDFS采用了多副本方式对数据进行冗余存储，一个数据块多个副本会分布到不同数据节点上
- 如图，黄色数据块被分别存放到数据节点**#1**和**#3**上，红色数据块被存放在数据节点**#1**、**#2**和**#3**上。多副本方式具有以下优点：
  - 加快数据传输速度
  - 容易检查数据错误
  - 保证数据可靠性



2.3 HDFS存储原理

b) 数据存取策略 -- 存放

- 第1个副本：放置在上传文件的数据节点；如果是集群外提交，则随机挑选一台磁盘不太满、CPU不太忙的节点
- 第2个副本：放置在与第1个副本不同的机架的节点上
- 第3个副本：与第2个副本相同机架的其他节点上
- 更多副本：随机节点



背后的原理是什么？

传输速率：节点内部>相同机架不同节点>同一集群不同机架上的节点>不同集群的节点

2.3 HDFS存储原理

b) 数据存取策略 -- 读取

- HDFS提供了一个API可以确定一个数据节点所属的机架ID，客户端也可以调用API获取自己所属的机架ID

- 当客户端读取数据时，从名称节点获得数据块不同副本的存放位置列表，列表中包含了副本所在的数据节点，可以调用API来确定客户端和这些数据节点所属的机架ID，当发现某个数据块副本对应的机架ID和客户端对应的机架ID相同时，就优先选择该副本读取数据，如果没有发现，就随机选择一个副本读取数据

2.3 HDFS存储原理

c) 数据错误和恢复

HDFS具有较高的容错性，可以兼容廉价的硬件，主要包括以下几种情形：名称节点出错、数据节点出错和数据出错。

名称节点出错

- 名称节点保存所有元数据信息，其中，最核心是FSImage和Edit logs，如果这两个文件发生损坏，整个HDFS实例将失效。
- HDFS设置备份机制，核心文件同步复制到Standby NameNode上。

数据节点出错

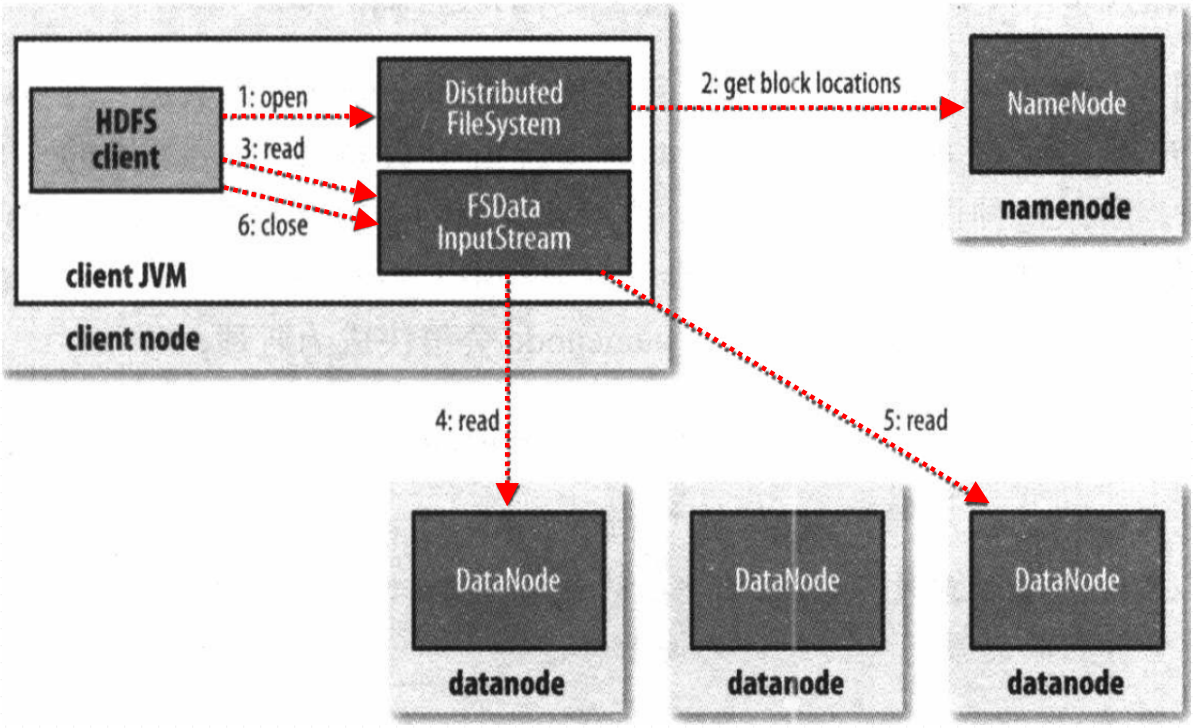
- 当数据节点发生故障时，名称节点无法收到一些数据节点心跳信息，会标记为“宕机”。
- 名称节点会定期检查这种情况，一旦发现某数据块的副本数量小于冗余因子，会启动数据冗余复制，生成新副本。

数据出错

- 网络传输和磁盘错误等因素，都会造成数据错误。
- 客户端在读取到数据后，会采用md5和sha1对数据块进行校验，以确定读取到正确的数据。

2.4 HDFS文件读写逻辑——文件读取

1. 客户端调用DistributedFileSystem对象的open()函数打开希望读取的文件；
2. DistributedFileSystem对象通过远程过程调用(RPC)调用namenode，确定文件起始块的位置。对于每一个块，namenode返回存有该块副本的datanode地址，并根据距离排序；
3. DistributedFileSystem类返回一个FSDataInput-Stream对象给客户端，客户端对这个输入流调用read()方法从最近的第一个块所在的datanode读取数据流；
4. 达到块末端时，关闭与该datanode的连接，寻找下一个最佳datanode继续调用read()方法读取数据。
5. 读取完成后，调用close()方法关闭输入流。



2.4 HDFS文件读写逻辑——文件读取中的错误处理

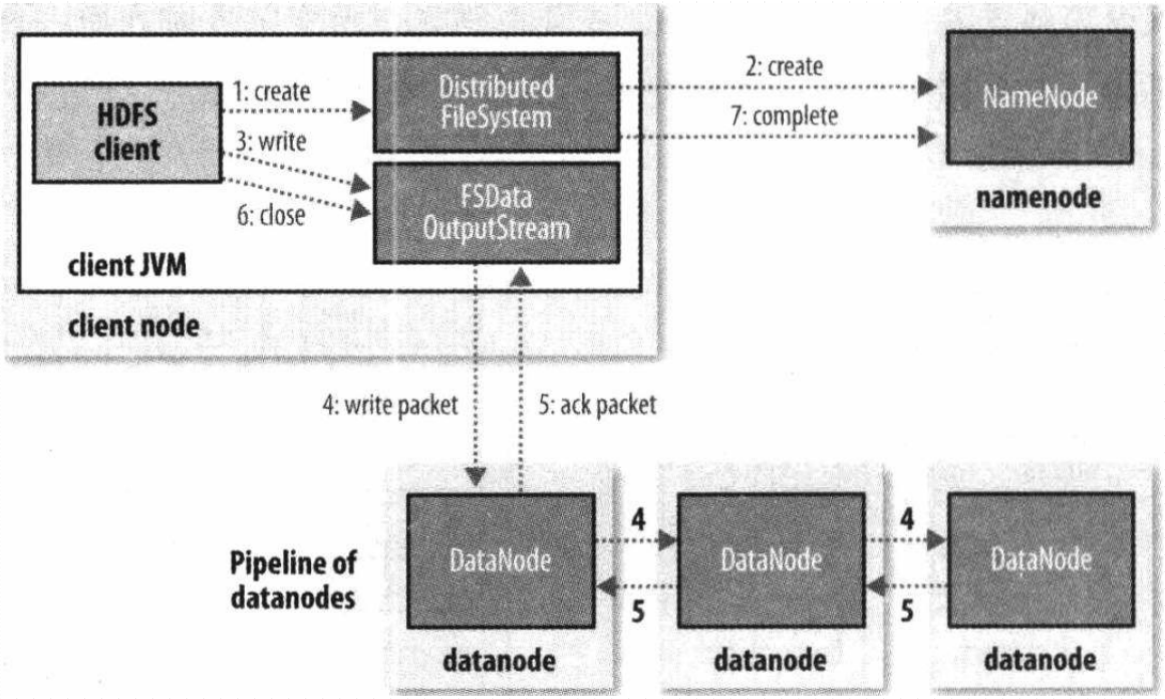
- 若DFSInputStream与datanode通信遇到错误，会尝试从这个块的另一个**最邻近**的datanode读取数据，并记住故障节点，避免以后反复读取该节点上后续的块。
- DFSInputStream也会通过校验和确认datanode发过来的数据是否完整，若发现损坏，DFSInputStream会尝试从其他datanode读取副本，并把损坏的块通知给namenode。
- NameNode只需响应块位置请求，无需相应数据请求，提升了系统的可扩展性。

关于datanode距离的定义（基于拓扑）

类型	距离
同一节点上的进程	0
同一机架上的不同节点	2
同一数据中心中不同机架上的节点	4
不同数据中心中的节点	6

2.4 HDFS文件读写逻辑——文件写入

1. 客户端调用open()函数新建文件（步骤1）；
2. DistributedFileSystem对namenode创建一个RPC，在文件系统的命名空间新建一个文件，此时还未写入数据块（步骤2）；
3. NameNode执行各种检查以确保该文件不存在且客户端具有相应权限，并返回一个FSDataOutputStream对象，负责与datanode通信；
4. FSDataOutputStream 将客户端写入的数据分成一个个数据包，并写入内部队列，DataStreamer处理数据队列，挑选出一组datanode，据此要求namenode分配新的数据块，这组数据块构成一个管线，然后通过管线依次将数据包写入到datanode上；
5. 写入成功后，返回确认信息，最后通过close方法结束。



2.5 HDFS常用命令行

命令行示例	解释
hadoop fs -help	获取每个命令的详细帮助信息
hadoop namenode -format	格式化一个新的分布式文件系统，谨慎使用
hadoop fs -mkdir -p /data/input	在HDFS上创建一个目录/data/input。-p会创建路径中各级父目录（如果没有的话）
hadoop fs -put aaa.txt /data/input	将当前目录（本地文件系统）下的aaa.txt 文件复制（上传）到HDFS上。源路径可以是多个
hadoop fs -get {源路径} {本地路径}	HDFS上复制文件到本地文件系统
hadoop fs -ls /data/input	查看HDFS中/data/input目录下的文件 。加上-R递归显示。
hadoop fs -rm {路径}	HDFS中删除指定的文件。只删除非空目录和文件（递归删除用-rm -r）

更多命令查看HDFS命令手册

2.6 HDFS接口API

Hadoop是基于Java语言的，因此可以方便地使用Java API与HDFS文件系统进行交互。

- **org.apache.hadoop.fs.FileSystem抽象类**
  - HDFS文件系统的抽象类，定义了Hadoop中一个文件系统的客户端接口，其具体实现包括：fs.LocalFileSystem, hdfs.DistributedFileSystem, fs.ftp.FTPFileSystem等。
  - 示例：从HDFS文件系统中读取文件，并将内容输出到控制台。

```
InputStream in = null;
try {
        in = new URL("hdfs://localhost/user/tom/test.txt").openStream();
        IOUtils.copyBytes(in, System.out, 4096, false);
} finally {
        IOUtils.closeStream(in);
}
```