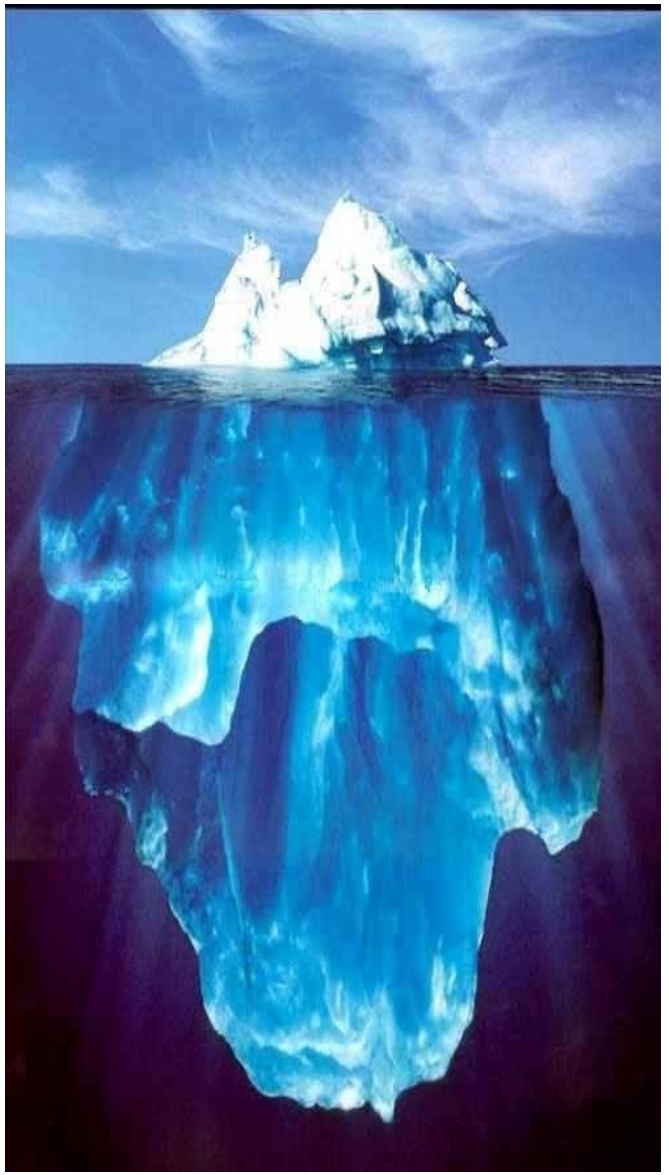
第三部分大数据分布式处理架构

内容

- Hadoop
- 分布式计算框架Mapreduce

Hadoop

- **HADOOP概述**
- **HDFS分布式文件系统**
- **YARN**



Hadoop简介

• 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
•Hadoop是基于Java语言开发的，具有很好的跨平台特性，并且可以部署在廉价的计算机集群中
•Hadoop的核心是分布式文件系统HDFS（Hadoop Distributed File System）和MapReduce
•Hadoop被公认为行业大数据标准开源软件，在分布式环境下提供了海量数据的处理能力
•几乎所有主流厂商都围绕Hadoop提供开发工具、开源软件、商业化工具和技术服务，如谷歌、雅虎、微软、思科、淘宝等，都支持Hadoop

Hadoop发展简史

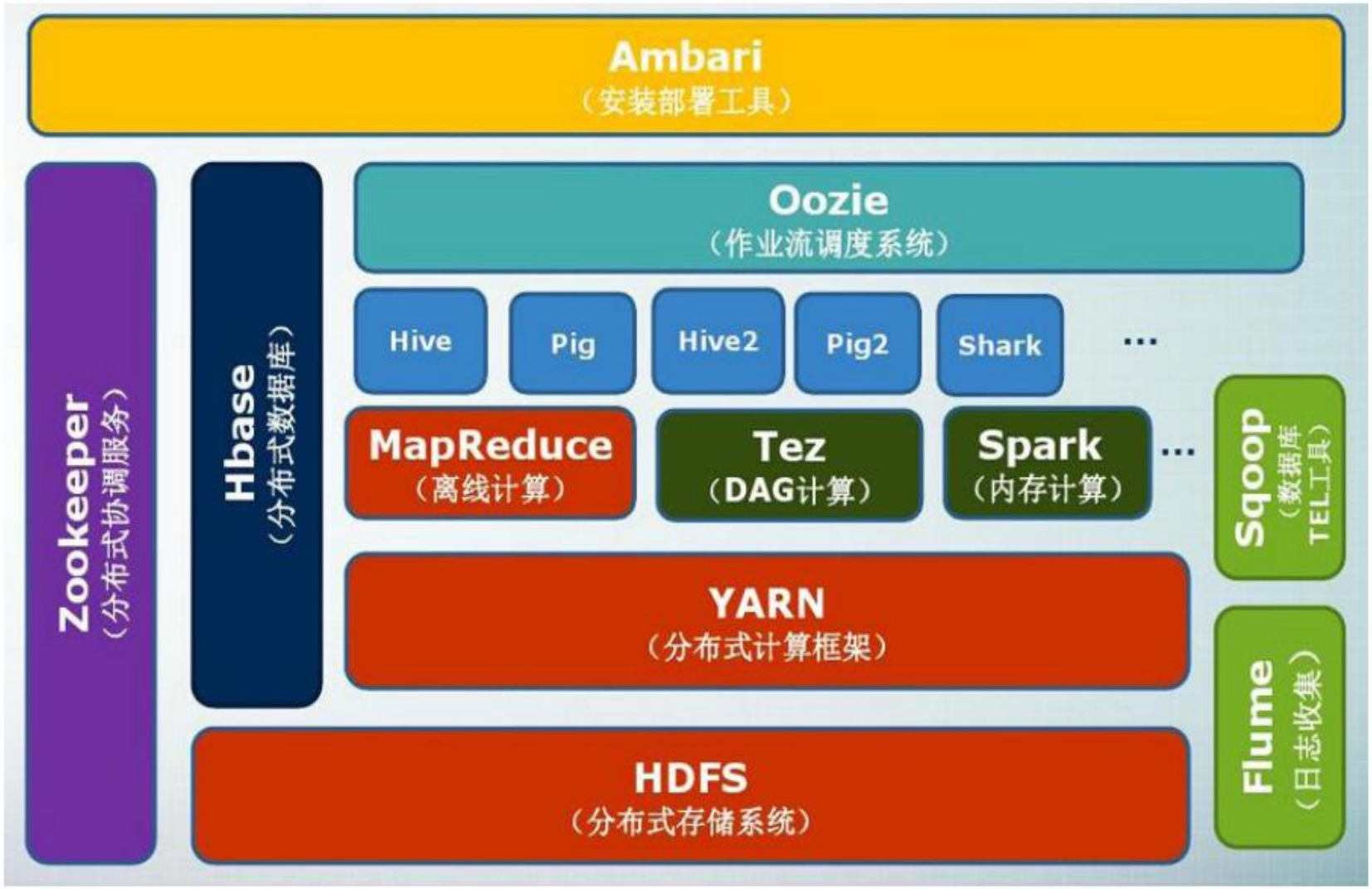
- 到了2006年2月，Nutch中的NDFS和MapReduce开始独立出来，成为Lucene项目的一个子项目，称为Hadoop，同时，Doug Cutting加盟雅虎
- 2008年1月，Hadoop正式成为Apache顶级项目，Hadoop也逐渐开始被雅虎之外的其他公司使用
- 2008年4月，Hadoop打破世界纪录，成为最快排序1TB数据的系统，它采用一个由910个节点构成的集群进行运算，排序时间只用了209秒
- 在2009年5月，Hadoop更是把1TB数据排序时间缩短到62秒。Hadoop从此名声大震，迅速发展成为大数据时代最具影响力的开源分布式开发平台，并成为事实上的大数据处理标准

Hadoop的应用现状

- 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
- 2007年，雅虎在Sunnyvale总部建立了M45——一个包含了4000个处理器和1.5PB容量的Hadoop集群系统
- Facebook作为全球知名的社交网站，Hadoop是非常理想的选择，Facebook主要将Hadoop平台用于日志处理、推荐系统和数据仓库等方面
- 国内采用Hadoop的公司主要有百度、淘宝、网易、华为、中国移动等，其中，淘宝的Hadoop集群比较大

Hadoop生态圈的结构

Hadoop的项目结构不断丰富发展，已经形成一个丰富的Hadoop生态系统



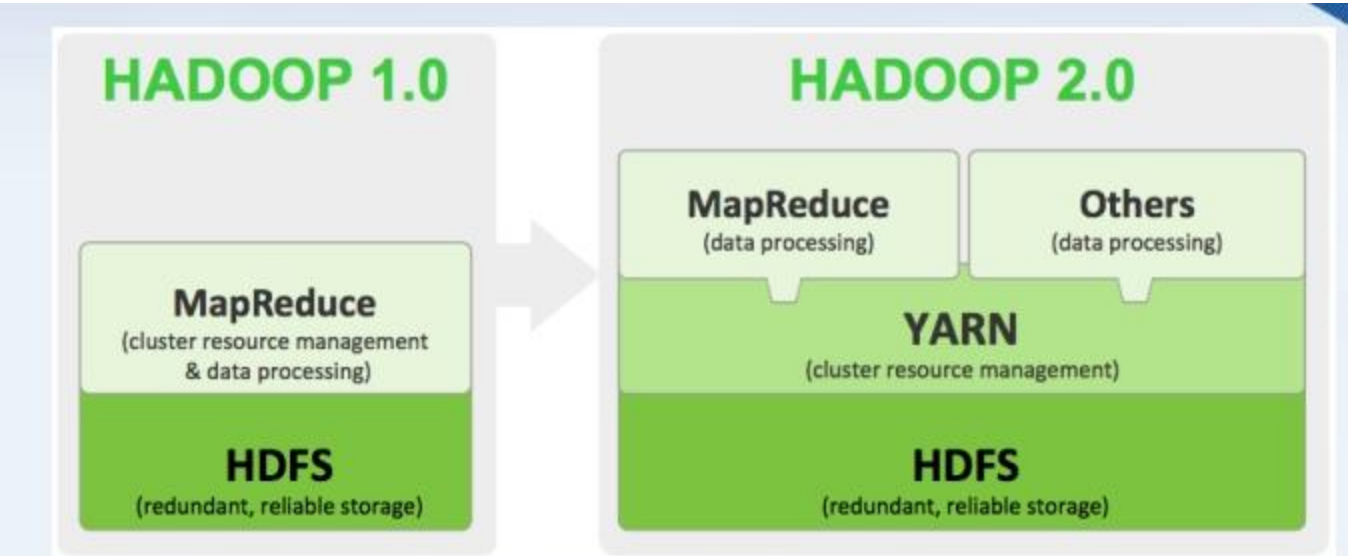
Hadoop项目结构

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	运行在YARN之上的下一代Hadoop查询处理框架
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
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，可以处理消费者规模的网站中的所有动作流数据
Spark	类似于Hadoop MapReduce的通用并行框架

Apache Hadoop版本演变

- Apache Hadoop版本分为三代，我们将第一代Hadoop称为Hadoop 1.0，第二代Hadoop称为Hadoop 2.0
- 第一代Hadoop包含三个大版本，分别是0.20.x，0.21.x和0.22.x，其中，0.20.x最后演化成1.0.x，变成了稳定版，而0.21.x和0.22.x则增加了NameNode HA等新的重大特性
- 第二代Hadoop包含两个版本，分别是0.23.x和2.x，它们完全不同于Hadoop 1.0，是一套全新的架构，均包含HDFS Federation和YARN两个系统，相比于0.23.x，2.x增加了NameNode HA和Wire-compatibility两个重大特性
- 第三代Hadoop。。。。。最新版本

Apache Hadoop版本演变



- ✓ 由HDFS、MapReduce和YARN三个分支构成；
- ✓ HDFS：NN Federation、HA；
- ✓ MapReduce：运行在YARN上的MR；

Hadoop各种版本

- Apache Hadoop
- Hortonworks
- Cloudera（CDH：Cloudera Distribution Hadoop）
- MapR
- ……

选择 Hadoop版本的考虑因素：
•是否开源（即是否免费）
•是否有稳定版
•是否经实践检验
•是否有强大的社区支持

Hadoop各种版本

厂商名称	开放性	易用性	平台功能	性能	本地支持	总体评价
apache	完全开源、Hadoop就是托管在apache社区里面	安装: 2 使用: 2 维护: 2	Apache是标准的Hadoop平台，所有厂商都是在apache的平台上面进行改进	2	没有	2
cloudera	与Apache功能同步，部分代码开源	安装: 5 使用: 5 维护: 5	有自主研发的产品如：impala、navigator等	4.5	2014年刚进入中国，上海	4.5
hortonworks	与apache功能同步，也是完全开源	安装: 4.5 使用: 5 维护: 5	是apache hadoop平台的最大贡献者，如Tez	4.5	没有	4.5
MapR	在apache的hadoop版本上面修改很多	安装: 4.5 使用: 5 维护: 5	在apache平台上面优化很多、从而形成自己的产品	5	没有	3.5
星环	核心组件与apache同步、底层的优化比较多、完全封闭的一个平台	安装: 5 使用: 4 维护: 4	有自主的Hadoop产品如 Inceptor、Hyperbase	4	本地厂商	4

Hadoop的安装与使用

- 选择Hadoop版本
- 选择运行模式
- 安装操作系统-Linux
- 安装Hadoop
- 配置
- 运行并查错

选择Hadoop发行版本（免费）

- Apache（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”）

CDH版本

- 最稳定全面，生产应用最多
- 下载http://archive.cloudera.com/cdh5/cdh/5/

运行模式

- 独立式
  - Hadoop运行所有的东西在无后台的单独的JVM中， 单进程单机模式
- 伪分布式
  - Hadoop做为后台应用运行在一台机器，模拟小集群
- 全分布式
  - Hadoop做为后台应用运行真实的集群多台电脑中

安装linux系统

- Hadoop最适合的OS
- Centos
- Ubuntu
- 虚拟机or真实linux平台

Hadoop的安装与使用（单机/伪分布式）

Hadoop基本安装配置主要包括以下几个步骤：

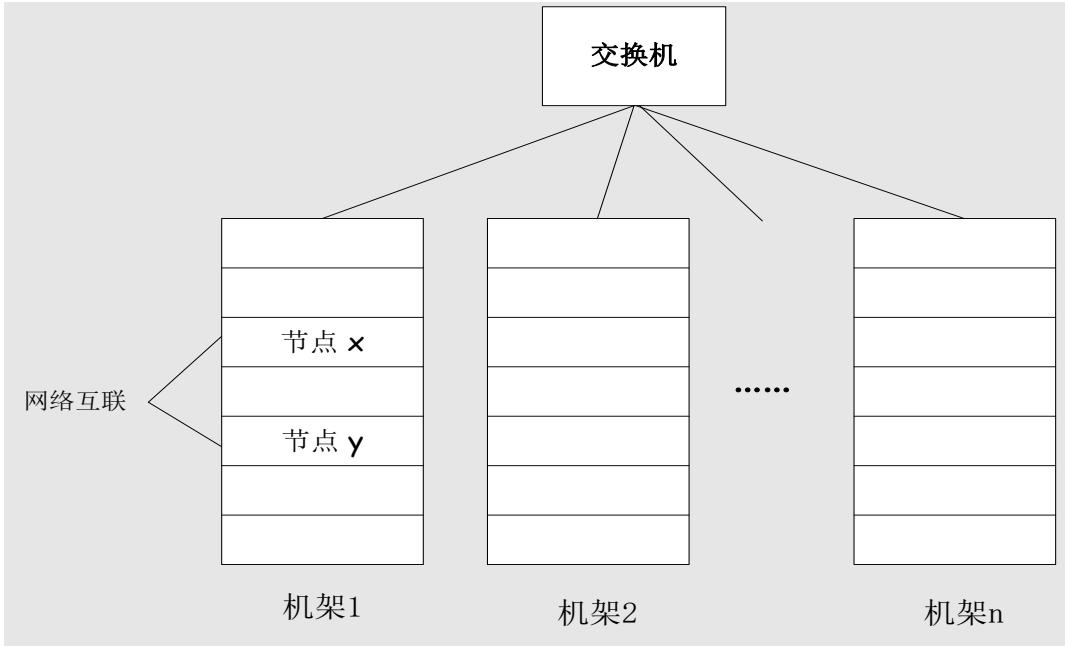
- 创建Hadoop用户
- SSH登录权限设置
- 安装Java环境
- 伪分布式安装配置
- 格式化
- 启动运行并验证

Hadoop集群中有哪些节点类型

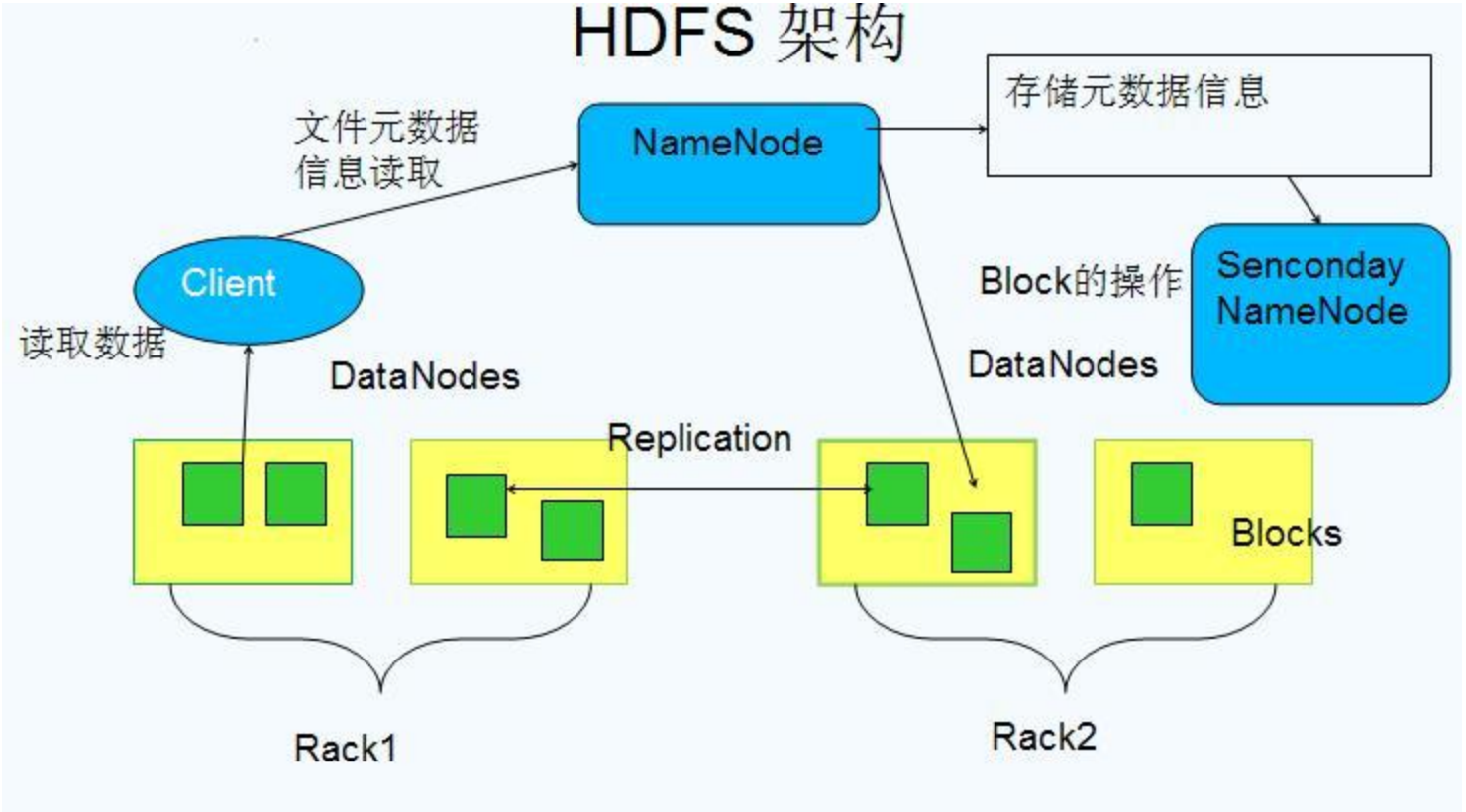
- Hadoop框架中最核心的设计是为海量数据提供存储的HDFS和对数据进行计算的MapReduce
- MapReduce的作业主要包括：（1）从磁盘或从网络读取数据，即IO密集工作；（2）计算数据，即CPU密集工作
- Hadoop集群的整体性能取决于CPU、内存、网络以及存储之间的性能平衡。因此运营团队在选择机器配置时要针对不同的工作节点选择合适硬件类型
- 一个基本的Hadoop集群中的节点主要有
  - NameNode：负责协调集群中的数据存储
  - DataNode：存储被拆分的数据块
  - JobTracker：协调数据计算任务（旧版）
  - TaskTracker：负责执行由JobTracker指派的任务（旧版）
  - SecondaryNameNode：帮助NameNode收集文件系统运行的状态信息，存储元数据

HDFS分布式文件系统

- 分布式文件系统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
- 与之前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，都是由普通硬件构成的，这就大大降低了硬件上的开销

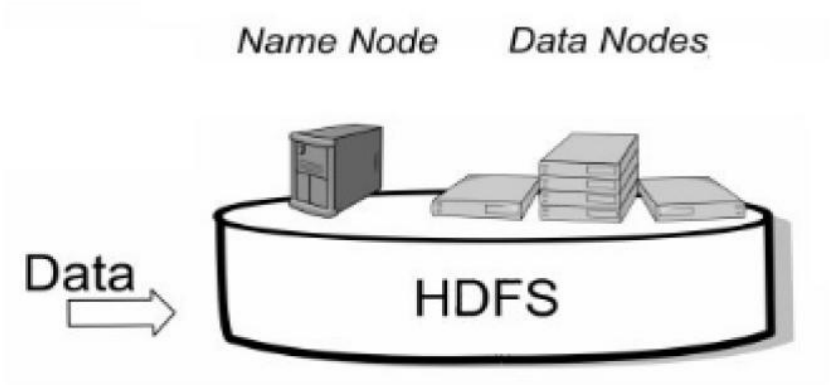


HDFS架构


HDFS 架构
文件元数据
信息读取
NameNode
存储元数据信息
Client
读取数据
Block的操作
Senconday
NameNode
DataNodes
DataNodes
Replication
Blocks
Rack1
Rack2

名称节点和数据节点

HDFS主要组件的功能



metadata

File.txt=
Blk A:
DN1, DN5, DN6

Blk B:
DN7, DN1, DN2

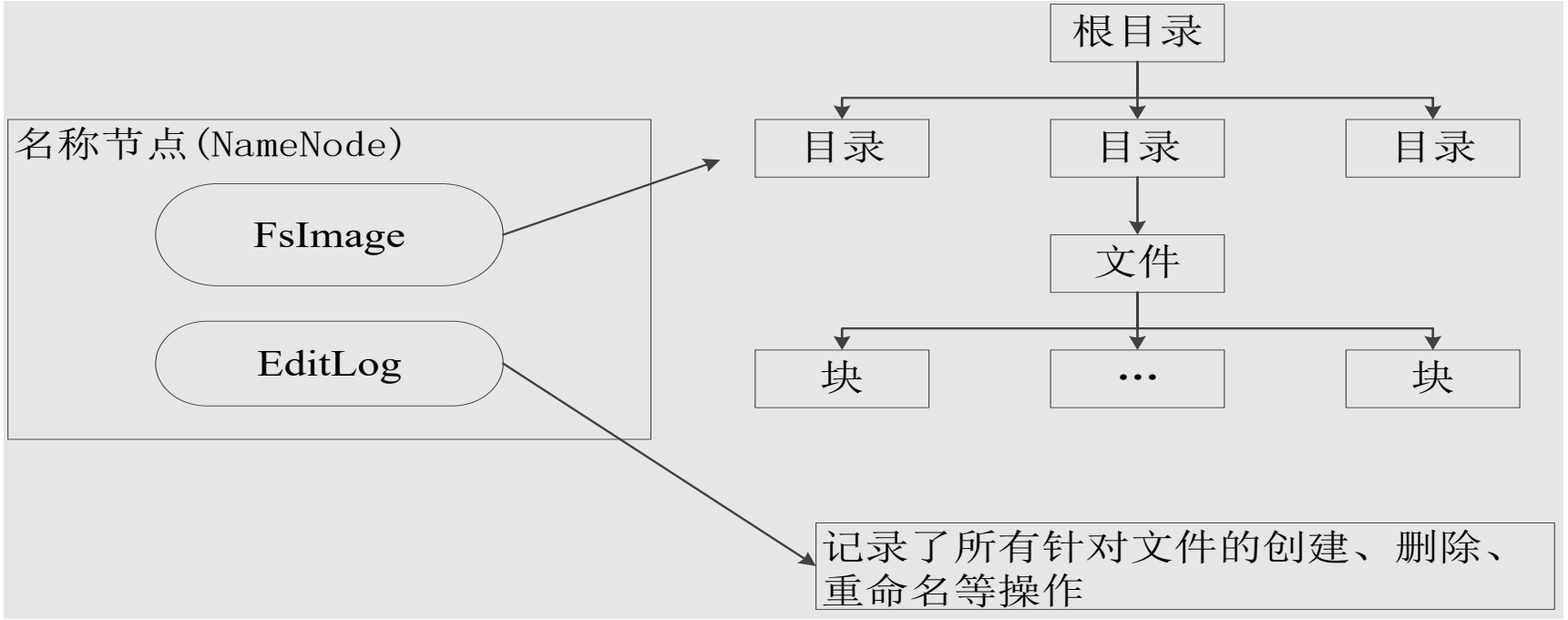
Blk C:
DN5, DN8,DN9

NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block， datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

名称节点数据结构

名称节点的数据结构

- 在HDFS中，名称节点（NameNode）负责管理分布式文件系统的命名空间（Namespace），保存了两个核心的数据结构，即FsImage和EditLog
  - FsImage用于维护文件系统树以及文件树中所有的文件和文件夹的元数据
  - 操作日志文件EditLog中记录了所有针对文件的创建、删除、重命名等操作
- 名称节点记录了每个文件中各个块所在的数据节点的位置信息



Secondary NameNode

名称节点运行期间EditLog不断变大的问题

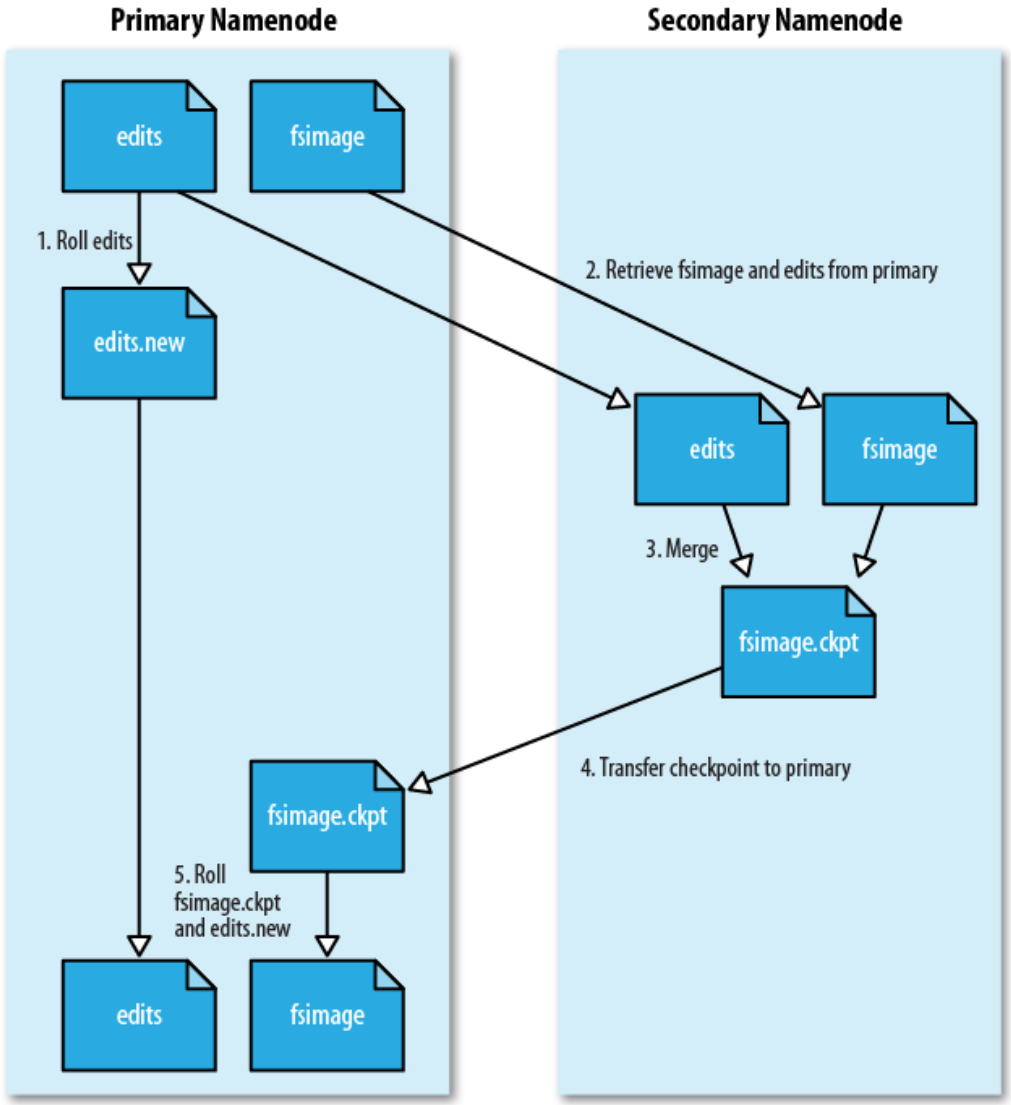
•在名称节点运行期间，HDFS的所有更新操作都是直接写到EditLog中，久而久之， EditLog文件将会变得很大

•虽然这对名称节点运行时候是没有什么明显影响的，但是，当名称节点重启的时候，名称节点需要先将FsImage里面的所有内容映像到内存中，然后再一条一条地执行EditLog中的记录，当EditLog文件非常大的时候，会导致名称节点启动操作非常慢，而在这段时间内HDFS系统处于安全模式，一直无法对外提供写操作，影响了用户的使用

如何解决？答案是：SecondaryNameNode第二名称节点

第二名称节点是HDFS架构中的一个组成部分，它是用来保存名称节点中对HDFS元数据信息的备份，并减少名称节点重启的时间。SecondaryNameNode一般是单独运行在一台机器上

SecondaryNameNode的工作流程



SecondaryNameNode的工作情况：

（1）SecondaryNameNode会定期和NameNode通信，请求其停止使用EditLog文件，暂时将新的写操作写到一个新的文件edit.new上来，这个操作是瞬间完成，上层写日志的函数完全感觉不到差别；

（2）SecondaryNameNode通过HTTP GET方式从NameNode上获取到FsImage和EditLog文件，并下载到本地的相应目录下；

（3）SecondaryNameNode将下载下来的FsImage载入到内存，然后一条一条地执行EditLog文件中的各项更新操作，使得内存中的FsImage保持最新；这个过程就是EditLog和FsImage文件合并；

（4）SecondaryNameNode执行完（3）操作之后，会通过post方式将新的FsImage文件发送到NameNode节点上

（5）NameNode将从SecondaryNameNode接收到的新的FsImage替换旧的FsImage文件，同时将edit.new替换EditLog文件，通过这个过程EditLog就变小了

数据节点

数据节点（DataNode）

•数据节点是分布式文件系统HDFS的工作节点，负责数据的存储和读取，会根据客户端或者是名称节点的调度来进行数据的存储和检索，并且向名称节点定期发送自己所存储的块的列表

•每个数据节点中的数据会被保存在各自节点的本地Linux文件系统中

Hadoop发展简史

Hadoop的标志

• Hadoop最初是由Apache Lucene项目的创始人Doug Cutting开发的文本搜索库。Hadoop源自始于2002年的Apache Nutch项目——一个开源的网络搜索引擎并且也是Lucene项目的一部分
• 在2004年，Nutch项目也模仿GFS开发了自己的分布式文件系统NDFS（Nutch Distributed File System），也就是HDFS的前身
• 2004年，谷歌公司又发表了另一篇具有深远影响的论文，阐述了MapReduce分布式编程思想
• 2005年，Nutch开源实现了谷歌的MapReduce

HDFS存储原理

- 冗余数据保存
- 数据存取策略
- 数据错误与恢复

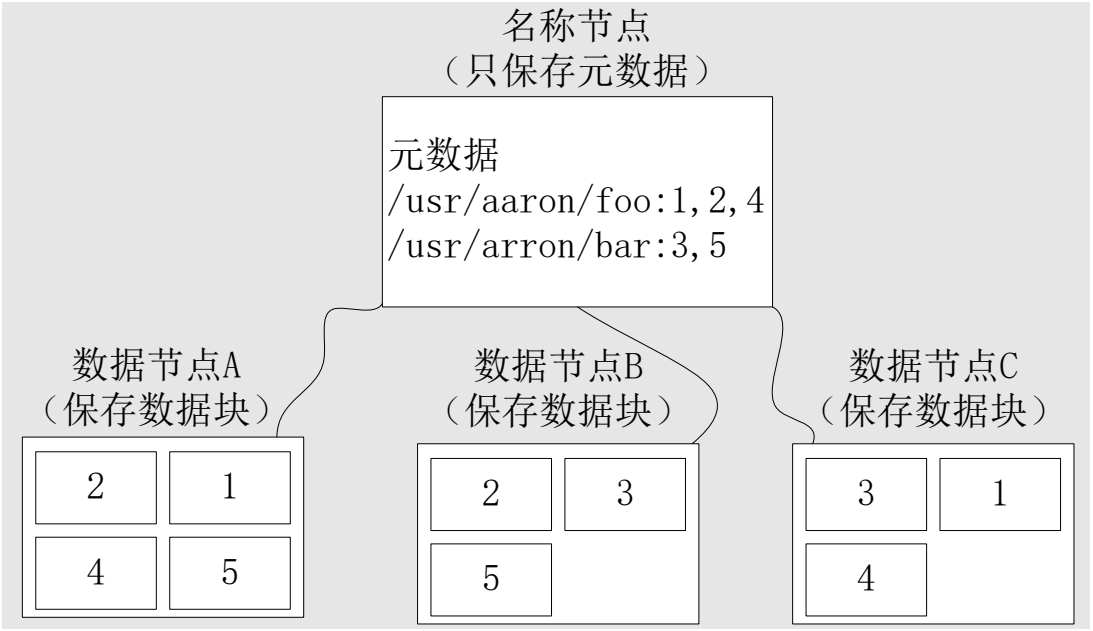
冗余数据保存

作为一个分布式文件系统，为了保证系统的容错性和可用性，HDFS采用了多副本方式对数据进行冗余存储，通常一个数据块的多个副本会被分布到不同的数据节点上，如图所示，数据块1被分别存放到数据节点A和C上，数据块2被存放在数据节点A和B上。这种多副本方式具有以下几个优点：

（1）加快数据传输速度

（2）容易检查数据错误

（3）保证数据可靠性

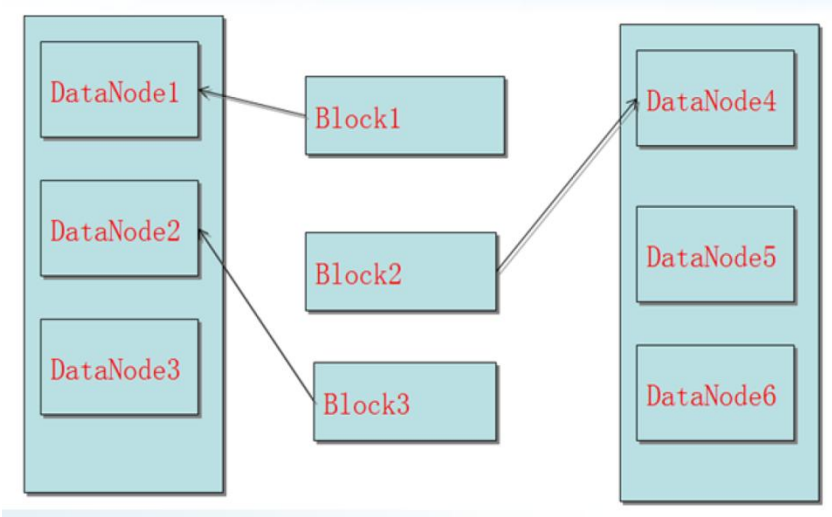


数据存取策略

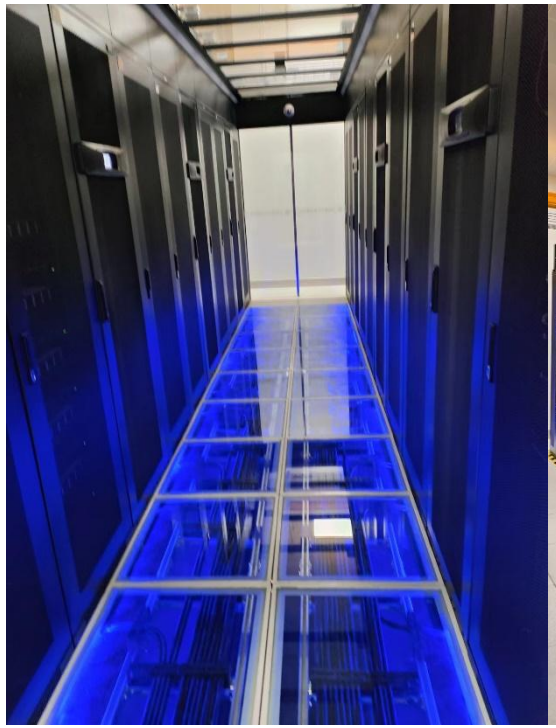
1.数据存放

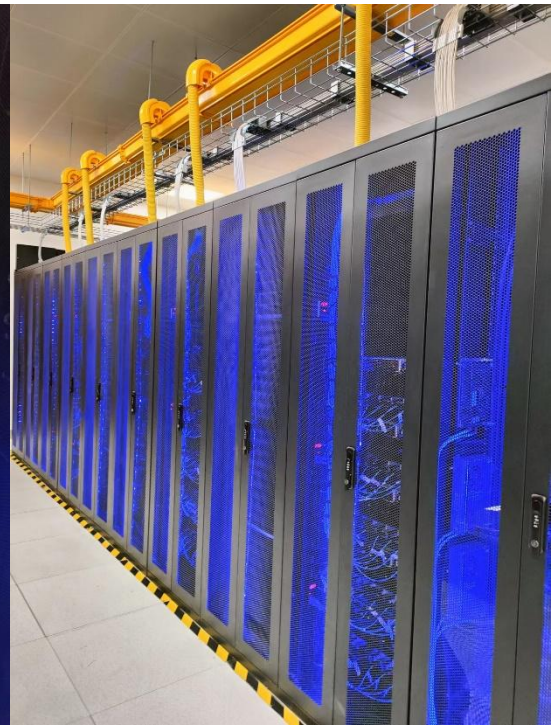
•第一个副本：放置在上传文件的数据节点；如果是集群外提交，则随机挑选一台磁盘不太满、CPU不太忙的节点
•第二个副本：放置在与第一个副本不同的机架的节点上
•第三个副本：与第一个副本相同机架的其他节点上
•更多副本：随机节点

Block的副本放置策略



数据中心





数据存取策略

2. 数据读取

•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

数据错误与恢复

HDFS具有较高的容错性，可以兼容廉价的硬件，它把硬件出错看作一种常态，而不是异常，并设计了相应的机制检测数据错误和进行自动恢复，主要包括以下几种情形：

1. **名称节点出错**
2. **数据节点出错**
3. **数据出错**

数据错误与恢复

名称节点出错

名称节点保存了所有的元数据信息，其中，最核心的两大数据结构是FsImage和Editlog，如果这两个文件发生损坏，那么整个HDFS实例将失效。因此，HDFS设置了备份机制，把这些核心文件同步复制到备份服务器SecondaryNameNode上。当名称节点出错时，就可以根据备份服务器SecondaryNameNode中的FsImage和Editlog数据进行恢复。

数据错误与恢复

数据节点出错

- 每个数据节点会定期向名称节点发送“心跳”信息，向名称节点报告自己的状态
- 当数据节点发生故障，或者网络发生断网时，名称节点就无法收到来自一些数据节点的心跳信息，这时，这些数据节点就会被标记为“宕机”，节点上面的所有数据都会被标记为“不可读”，名称节点不会再给它们发送任何I/O请求
- 这时，有可能出现一种情形，即由于一些数据节点的不可用，会导致一些数据块的副本数量小于冗余因子
- 名称节点会定期检查这种情况，一旦发现某个数据块的副本数量小于冗余因子，就会启动数据冗余复制，为它生成新的副本
- HDFS和其它分布式文件系统的最大区别就是可以调整冗余数据的位置

数据错误与恢复

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
- 在文件被创建时，客户端就会对每一个文件块进行信息摘录，并把这些信息写入到同一个路径的隐藏文件里面
- 当客户端读取文件的时候，会先读取该信息文件，然后，利用该信息文件对每个读取的数据块进行校验，如果校验出错，客户端就会请求到另外一个数据节点读取该文件块，并且向名称节点报告这个文件块有错误，名称节点会定期检查并且重新复制这个块

HDFS的文件操作

- shell命令， 类似linux命令
- JAVA　API编程接口

HDFS常用命令

Hadoop中有三种Shell命令方式：

- hadoop fs适用于任何不同的文件系统，比如本地文件系统和HDFS文件系统
- hadoop dfs 只能适用hadoop dfs的命令于HDFS文件系统
- hdfs dfs跟作用一样，也只能适用于HDFS文件系统

实例

- hdfs dfs -ls /
- mkdir(创建目录)
  - hdfs dfs-mkdir [-p]（p和Linux系统效果一样)
  - hdfs dfs -mkdir -p /test/test1/test/test2
- mv
  - hdfs dfs -mv URI [URI ...]
  - hdfs dfs –mv /output/CHANGES.txt/output/README.md/test/test2/
- put(上传)
  - hdfs dfs -put ...
  - hadoop dfs-put c.txt d.txt/test/test2/
- rm（删除）
  - hdfs dfs -rm [-f] [-r|-R] [-skipTrash] URI [URI ...] (f：是否确认 r:递归删除 skipTrash：直接删除)
  - hadoop dfs-rm-r/test/
- 更多命令示例，查看帮助 打命令： hadoop dfs

Java api

- 对分HDFS中的文件操作主要涉及一下几个类:
  - Configuration类：该类的对象封转了客户端或者服务器的配置。
  - FileSystem类：该类的对象是一个文件系统对象，可以用该对象的一些方法来对文件进行操作。FileSystem fs = FileSystem.get(conf);通过FileSystem的静态方法get获得该对象。
  - FSDataInputStream和FSDataOutputStream：这两个类是HDFS中的输入输出流。分别通过FileSystem的open方法和create方法获得。

java hadoop API示例

```
 //创建目录
public static void mkdir(String path) throws IOException{
Configuration conf = new Configuration();
 FileSystem fs = FileSystem.get(conf);
 Path srcPath = new Path(path);
 boolean isok = fs.mkdirs(srcPath);
 if(isok){
 System.out.println("create dir ok!");
  }else{
     System.out.println("create dir failure");
   }
 fs.close();
 }
    //读取文件的内容
   public static void readFile(String filePath) throws IOException{
   Configuration conf = new Configuration();
   FileSystem fs = FileSystem.get(conf);
   Path srcPath = new Path(filePath);
  InputStream in = null;
  try {
  in = fs.open(srcPath);
     IOUtils.copyBytes(in, System.out, 4096, false); //复制到标准输出流
  } finally {
      IOUtils.closeStream(in);
 } }
```

Hadoop和HDFS的局限与不足

Hadoop1.0的核心组件（仅指MapReduce和HDFS，不包括Hadoop生态系统内的Pig、Hive、HBase等其他组件），主要存在以下不足：

- 抽象层次低，需人工编码
- 表达能力有限
- 开发者自己管理作业（Job）之间的依赖关系
- 难以看到程序整体逻辑
- 执行迭代操作效率低
- 资源浪费（Map和Reduce分两阶段执行）
- 实时性差（适合批处理，不支持实时交互式）

针对Hadoop的改进与提升

Hadoop的优化与发展主要体现在两个方面：

- 一方面是Hadoop自身两大核心组件MapReduce和HDFS的架构设计改进

- 另一方面是Hadoop生态系统其它组件的不断丰富，加入了Pig、Tez、Spark和Kafka等新组件

针对Hadoop的改进与提升

不断完善的Hadoop生态系统

组件	功能	解决Hadoop中存在的问题
Pig	处理大规模数据的脚本语言，用户只需要编写几条简单的语句，系统会自动转换为MapReduce作业	抽象层次低，需要手工编写大量代码
Spark	基于内存的分布式并行编程框架，具有较高的实时性，并且较好支持迭代计算	延迟高，而且不适合执行迭代计算
Oozie	工作流和协作服务引擎，协调Hadoop上运行的不同任务	没有提供作业（Job）之间依赖关系管理机制，需要用户自己处理作业之间依赖关系
Tez	支持DAG作业的计算框架，对作业的操作进行重新分解和组合，形成一个大的DAG作业，减少不必要操作	不同的MapReduce任务之间存在重复操作，降低了效率
Kafka	分布式发布订阅消息系统，一般作为企业大数据分析平台的数据交换枢纽，不同类型的分布式系统可以统一接入到Kafka，实现和Hadoop各个组件之间的不同类型数据的实时高效交换	Hadoop生态系统中各个组件和其他产品之间缺乏统一的、高效的数据交换中介

针对Hadoop的改进与提升

Hadoop框架自身的改进：从1.0到2.0

组件	**Hadoop1.0的问题**	**Hadoop2.0的改进**
HDFS	单一名称节点，存在单点失效问题	设计了HDFS HA，提供名称节点热备机制
HDFS	单一命名空间，无法实现资源隔离	设计了HDFS Federation，管理多个命名空间
MapReduce	资源管理效率低	设计了新的资源管理框架YARN

HDFS2.0的新特性

HDFS HA
HDFS Federation

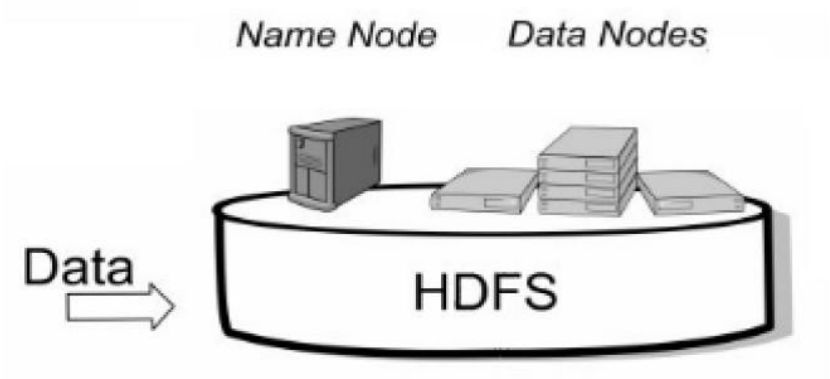
HDFS HA

HDFS1.0组件及其功能回顾

名称节点保存元数据：

（1）在磁盘上：FsImage和EditLog

（2）在内存中：映射信息，即文件包含哪些块，每个块存储在哪个数据节点



metadata

File.txt=
Blk A:
DN1, DN5, DN6

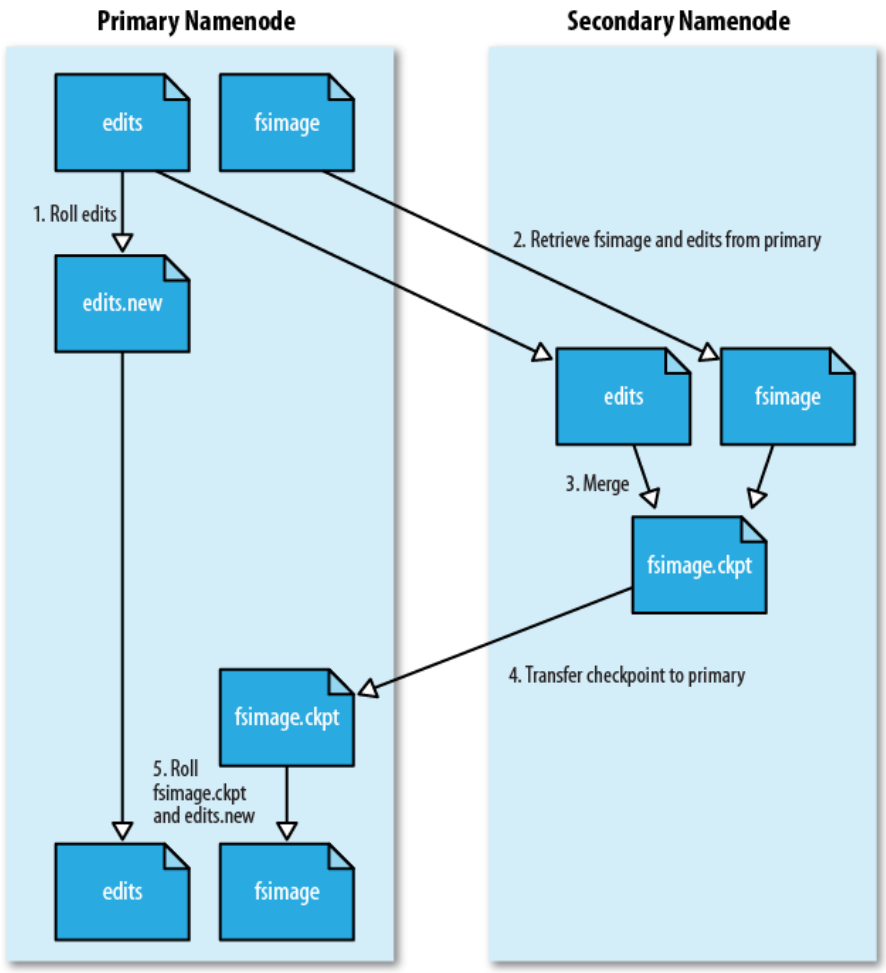
Blk B:
DN7, DN1, DN2

Blk C:
DN5, DN8,DN9

NameNode	DataNode
• 存储元数据	• 存储文件内容
•元数据保存在内存中	•文件内容保存在磁盘
• 保存文件,block， datanode之间的映射关系	•维护了block id到datanode本地文件的映射关系

HDFS HA

•HDFS 1.0存在单点故障问题
•第二名称节点（SecondaryNameNode）无法解决单点故障问题

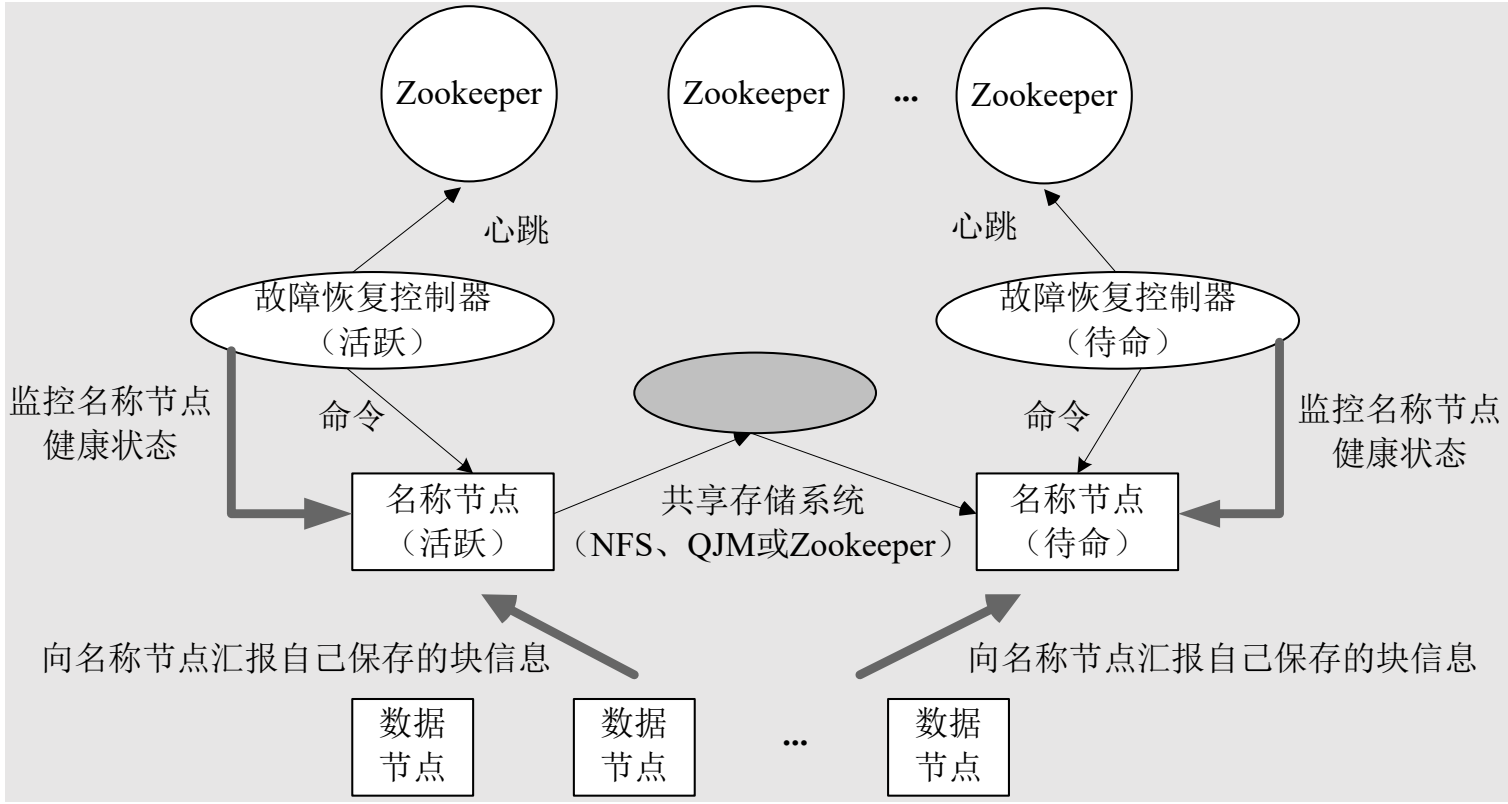


•SecondaryNameNode会定期和NameNode通信
•从NameNode上获取到FsImage和EditLog文件，并下载到本地的相应目录下
•执行EditLog和FsImage文件合并
•将新的FsImage文件发送到NameNode节点上
•NameNode使用新的FsImage和EditLog（缩小了）

第二名称节点用途：
•不是热备份
•主要是防止日志文件EditLog过大，导致名称节点失败恢复时消耗过多时间
•附带起到冷备份功能

HDFS HA

- HDFS HA（High Availability）是为了解决单点故障问题
- HA集群设置两个名称节点，“活跃（Active）”和“待命（Standby）”
- 两种名称节点的状态同步，可以借助于一个共享存储系统来实现
- 一旦活跃名称节点出现故障，就可以立即切换到待命名称节点
- Zookeeper确保一个名称节点在对外服务
- 名称节点维护映射信息，数据节点同时向两个名称节点汇报信息



构

HDFS Federation

1.HDFS1.0中存在的问题

- 单点故障问题
- 不可以水平扩展
- 系统整体性能受限于单个名称节点的吞吐量
- 单个名称节点难以提供不同程序之间的隔离性
- HDFS HA是热备份，提供高可用性，但是无法解决可扩展性、系统性能和隔离性

2.HDFS Federation的设计

- 在HDFS Federation中，设计了多个相互独立的名称节点，使得HDFS的命名服务能够水平扩展，这些名称节点分别进行各自命名空间和块的管理，相互之间是联盟（Federation）关系，不需要彼此协调。并且向后兼容
- HDFS Federation中，所有名称节点会共享底层的数据节点存储资源，数据节点向所有名称节点汇报
- 属于同一个命名空间的块构成一个“块池”

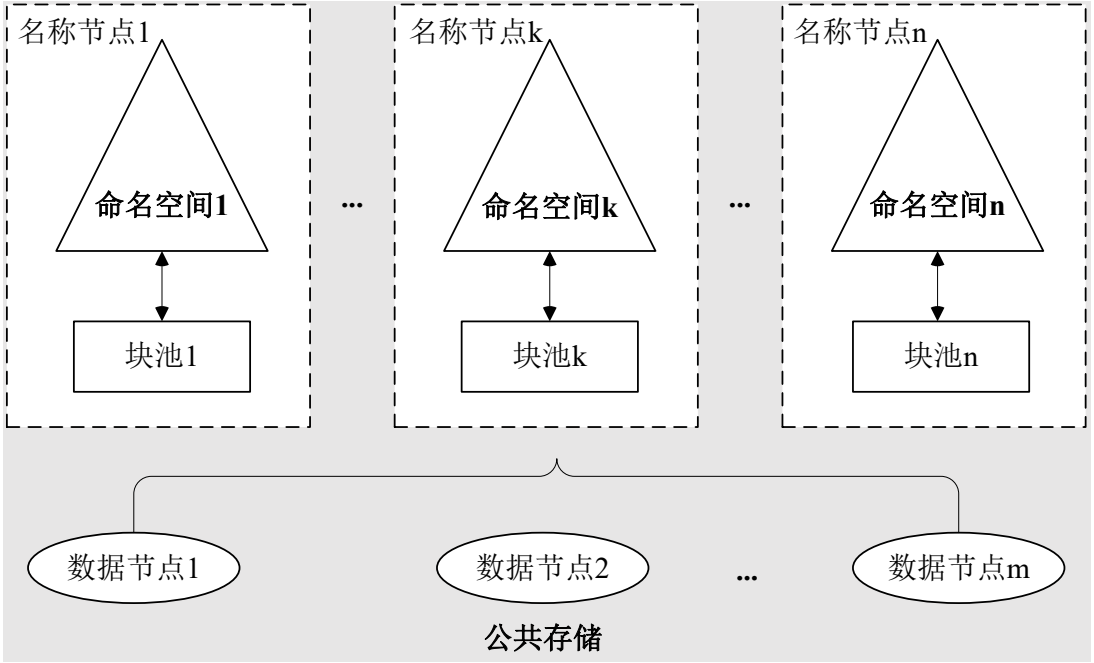


图 HDFS Federation架构

HDFS Federation

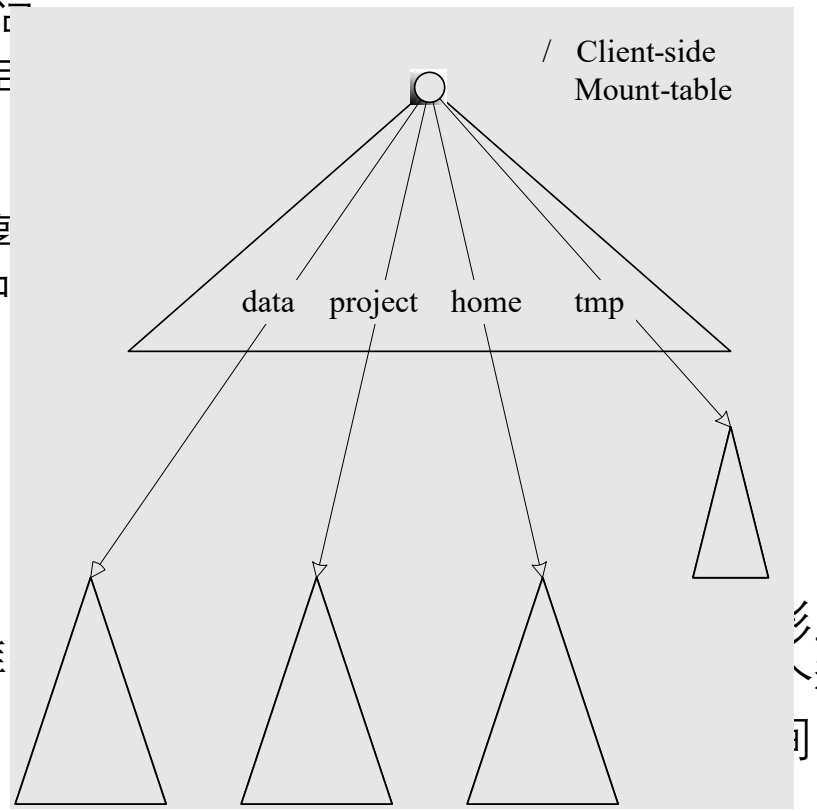
3. HDFS Federation的访问方式

•对于Federation中的多个命名空间，可以采用客户端挂载表（Client Side Mount Table）方式进行数据共享和访问

•客户可以访问不同的挂载点来访问不同

•把各个命名空间挂载到全局“挂载表”（mount-table）中，实现数据全局共享

•同样的命名空间挂载到个人的挂载表中就成为应用程序可见的命名空间



名称节点1维护的命名空间

彡三角形
个独立的
间

名称节点2维护的命名空间

HDFS Federation

4.HDFS Federation相对于HDFS1.0的优势

HDFS Federation设计可解决单名称节点存在的以下几个问题：

（1）HDFS集群扩展性。多个名称节点各自分管一部分目录，使得一个集群可以扩展到更多节点，不再像HDFS1.0中那样由于内存的限制制约文件存储数目

（2）性能更高效。多个名称节点管理不同的数据，且同时对外提供服务，将为用户提供更高的读写吞吐率

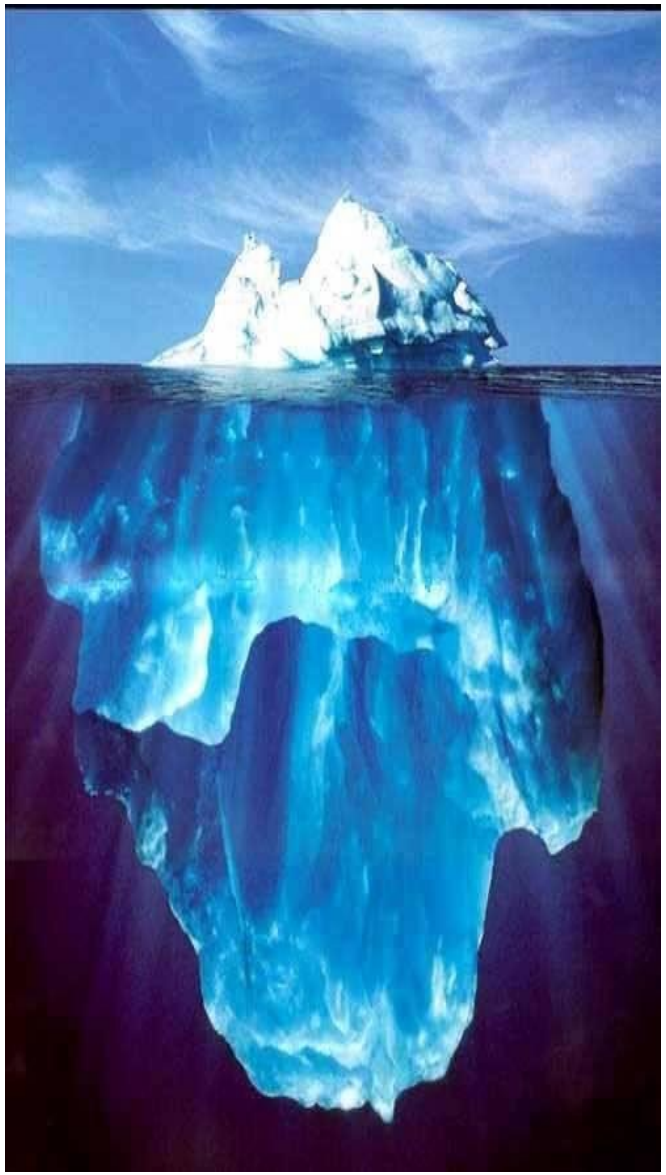
（3）良好的隔离性。用户可根据需要将不同业务数据交由不同名称节点管理，这样不同业务之间影响很小

大数据分布式计算框架

分而治之的智慧：MapReduce

提纲

- 概述
- MapReduce体系结构
- MapReduce工作流程
- 新一代资源调度器YARN
- 实例分析：WordCount
- MapReduce的具体应用
- MapReduce编程实践



概述

- 分布式并行编程
- MapReduce模型简介
- Map和Reduce函数

分布式并行编程

- “摩尔定律”， CPU性能大约每隔18个月翻一番
- 从2005年开始摩尔定律逐渐失效，需要处理的数据量快速增加，人们开始借助于分布式并行编程来提高程序性能
- 分布式程序运行在大规模计算机集群上，可以并行执行大规模数据处理任务，从而获得海量的计算能力
- 谷歌公司最先提出了分布式并行编程模型MapReduce，Hadoop MapReduce是它的开源实现，后者比前者使用门槛低很多

分布式并行编程

问题：在MapReduce出现之前，已经有像MPI这样非常成熟的并行计算框架了，那么为什么Google还需要MapReduce？MapReduce相较于传统的并行计算框架有什么优势？

	传统并行计算框架	MapReduce
集群架构/容错性	共享式(共享内存/共享存储)，容错性差	非共享式，容错性好
硬件/价格/扩展性	刀片服务器、高速网、SAN，价格贵，扩展性差	普通PC机，便宜，扩展性好
编程/学习难度	what-how，难	what，简单
适用场景	实时、细粒度计算、计算密集型	批处理、非实时、数据密集型

MapReduce模型简介

- MapReduce将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数：Map和Reduce
- 编程容易，不需要掌握分布式并行编程细节，也可以很容易把自己的程序运行在分布式系统上，完成海量数据的计算
- MapReduce采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理

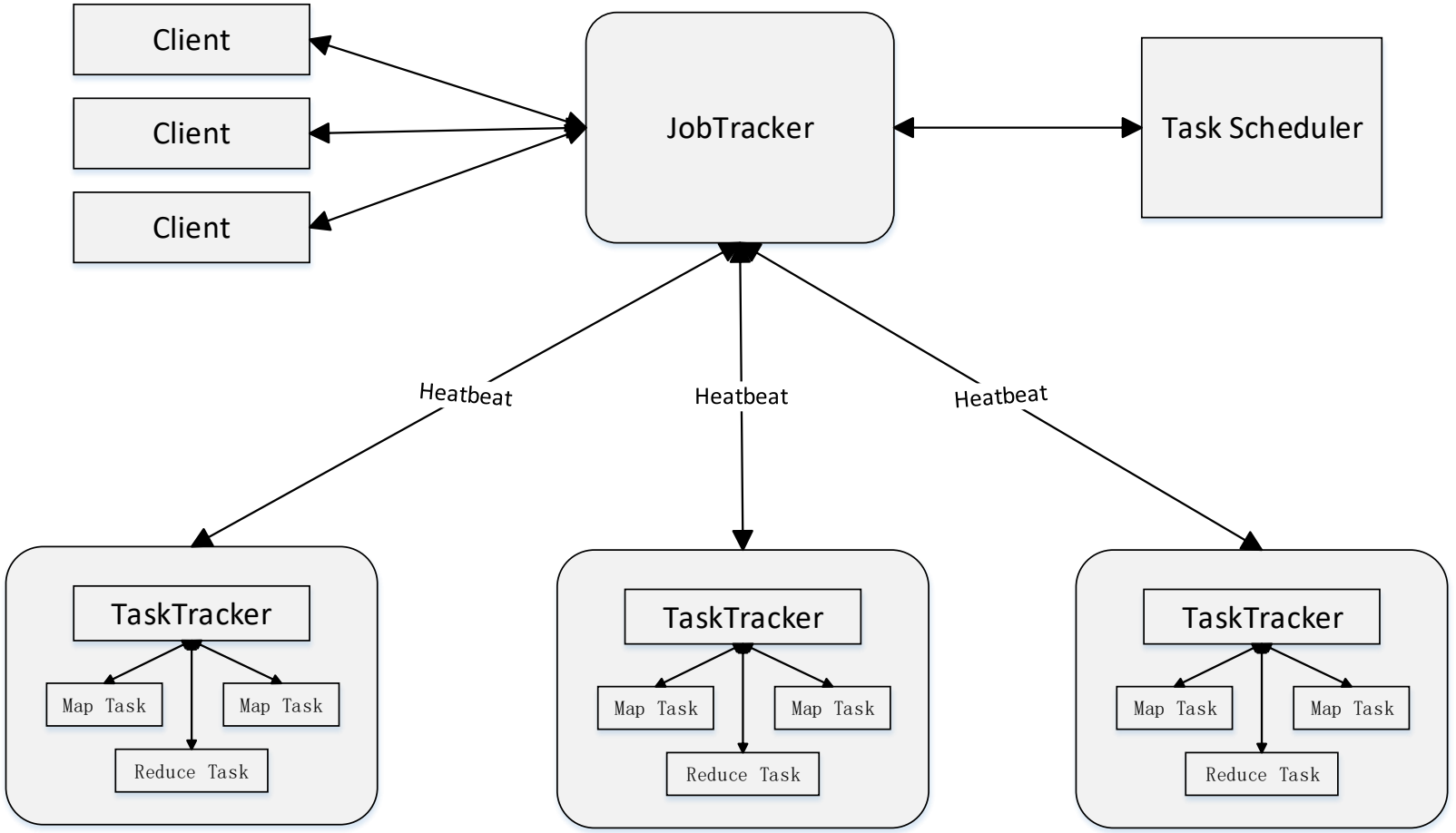
Map和Reduce函数

Map和Reduce

函数	输入	输出	说明
Map	<k_1,v$_1$> 如： <行号,"a b c">	List(<k_2,v_2>) 如： <"a",1> <"b",1> <"c",1>	1.将小数据集进一步解析成一批<key,value>对，输入Map函数中进行处理 2.每一个输入的<k_1,v_1>会输出一批<k_2,v_2>。<k_2,v_2>是计算的中间结果
Reduce	<k_2,List(v_2)> 如： <"a",<1,1,1>>	<k_3,v_3> <"a",3>	输入的中间结果<k_2,List(v_2)>中的List(v_2)表示是一批属于同一个k_2的value

MapReduce的体系结构

MapReduce体系结构主要由四个部分组成，分别是：Client、JobTracker、TaskTracker以及Task



MapReduce的体系结构

MapReduce主要有以下4个部分组成：

1）Client

•用户编写的MapReduce程序通过Client提交到JobTracker端

•用户可通过Client提供的一些接口查看作业运行状态

2）JobTracker

•JobTracker负责资源监控和作业调度

•JobTracker 监控所有TaskTracker与Job的健康状况，一旦发现失败，就将相应的任务转移到其他节点

•JobTracker 会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器（TaskScheduler），而调度器会在资源出现空闲时，选择合适的任务去使用这些资源

MapReduce的体系结构

3）TaskTracker

•TaskTracker 会周期性地通过“心跳”将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）

•TaskTracker 使用“slot”等量划分本节点上的资源量（CPU、内存等）。一个Task 获取到一个slot 后才有机会运行，而Hadoop调度器的作用就是将各个TaskTracker上的空闲slot分配给Task使用。slot 分为Map slot 和Reduce slot 两种，分别供MapTask 和Reduce Task 使用

4）Task

Task 分为Map Task 和Reduce Task 两种，均由TaskTracker 启动

新一代资源管理调度框架YARN

- MapReduce1.0的缺陷
- YARN设计思路
- YARN体系结构

MapReduce1.0的缺陷

（1）存在单点故障
（2）JobTracker“大包大揽”导致任务过重（任务多时内存开销大，上限4000节点）
（3）容易出现内存溢出（分配资源只考虑MapReduce任务数，不考虑CPU、内存）
（4）资源划分不合理（强制划分为slot ，包括Map slot和Reduce slot）

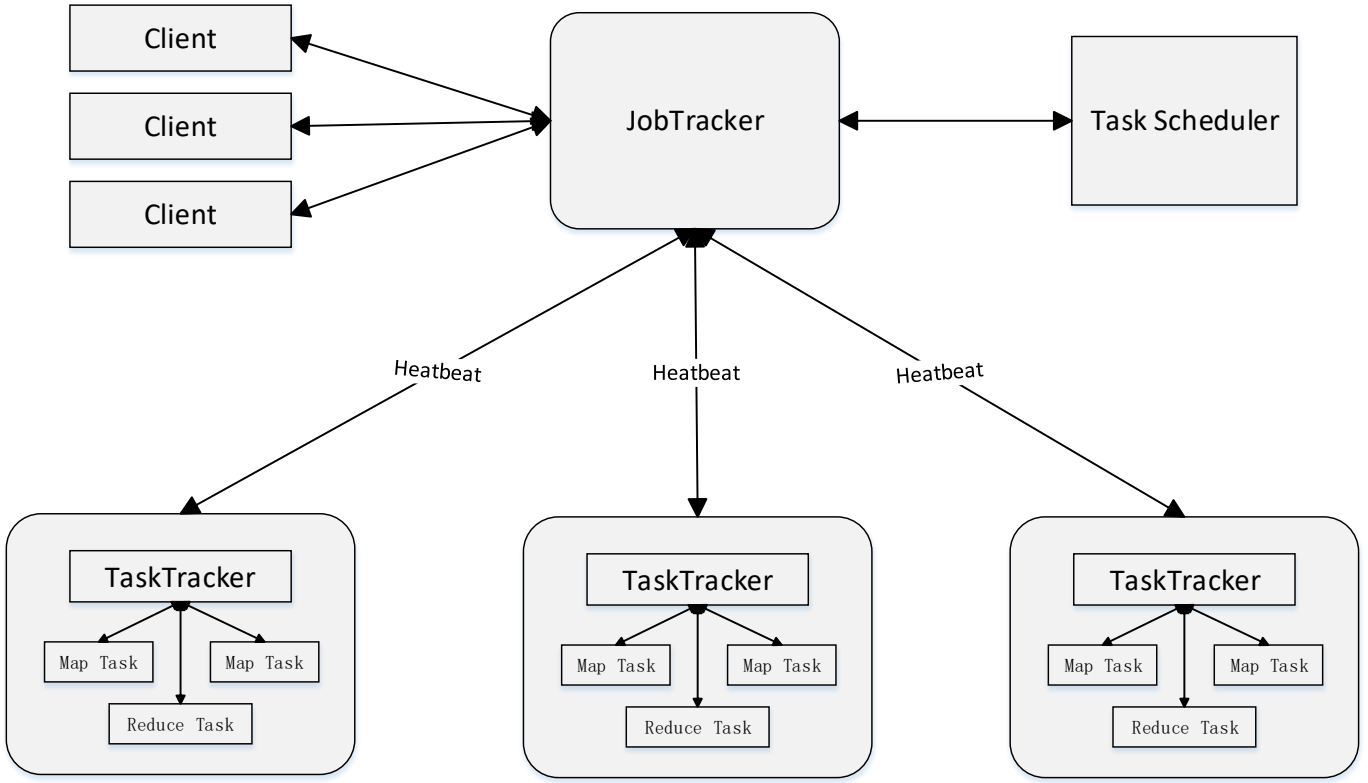
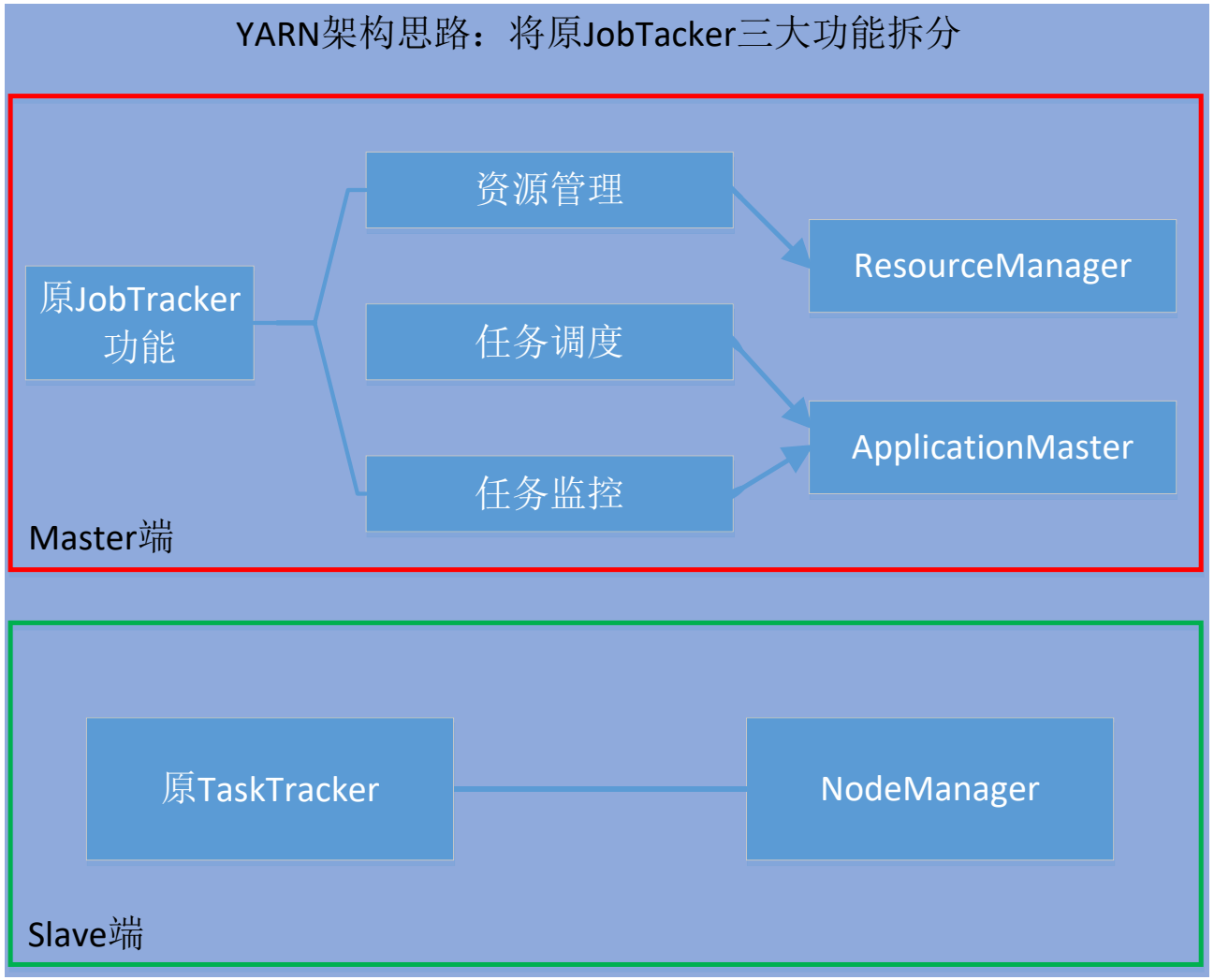


图 MapReduce1.0体系结构

YARN设计思路



- MapReduce1.0既是一个计算框架，也是一个资源管理调度框架
- 到了Hadoop2.0以后，MapReduce1.0中的资源管理调度功能，被单独分离出来形成了YARN，它是一个纯粹的资源管理调度框架，而不是一个计算框架
- 被剥离了资源管理调度功能的MapReduce 框架就变成了MapReduce2.0，它是运行在YARN之上的一个纯粹的计算框架，不再自己负责资源调度管理服务，而是由YARN为其提供资源管理调度服务

YARN体系结构

ResourceManager

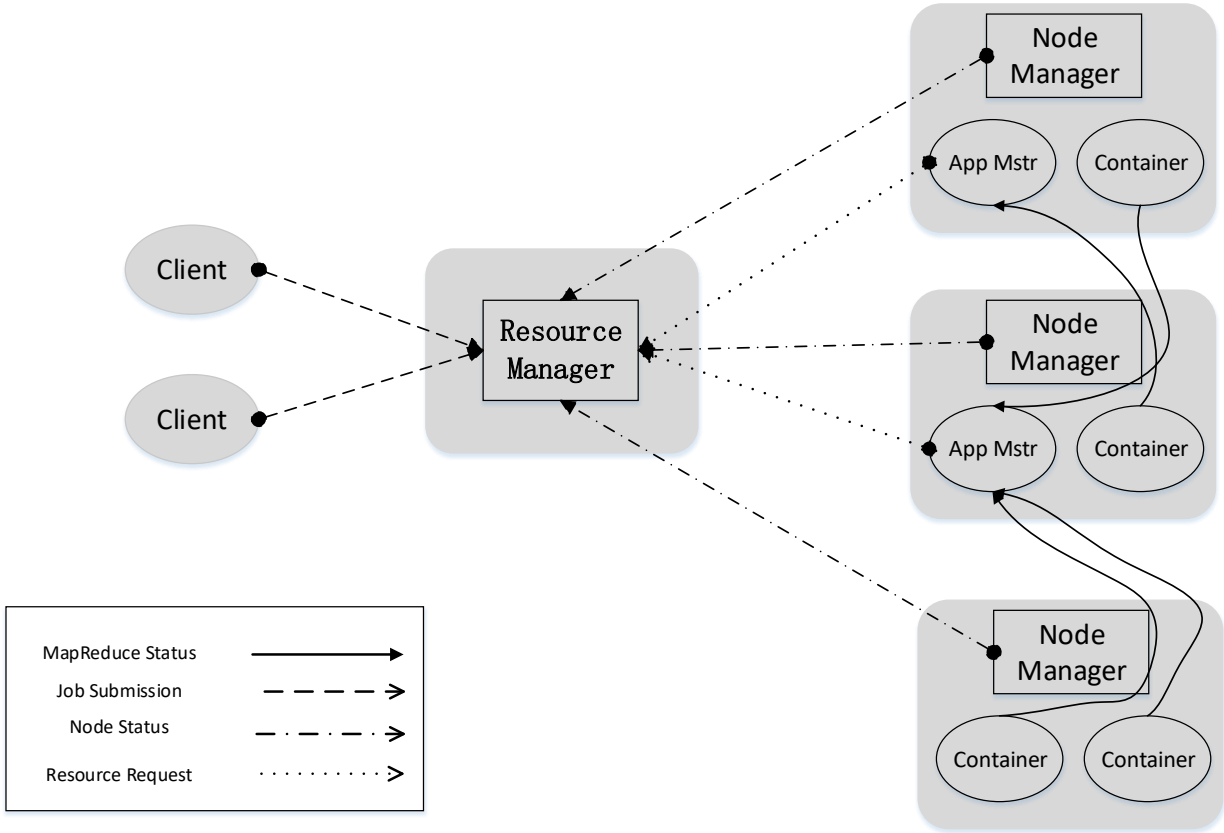
- 处理客户端请求
- 启动/监控ApplicationMaster
- 监控NodeManager
- 资源分配与调度

NodeManager

- 单个节点上的资源管理
- 处理来自ResourceManger的命令
- 处理来自ApplicationMaster的命令

ApplicationMaster

- 为应用程序申请资源，并分配给内部任务
- 任务调度、监控与容错



YARN体系结构

ResourceManager

•ResourceManager（RM）是一个全局的资源管理器，负责整个系统的资源管理和分配，主要包括两个组件，即调度器（Scheduler）和应用程序管理器（Applications Manager）

•调度器接收来自ApplicationMaster的应用程序资源请求，把集群中的资源以“容器”的形式分配给提出申请的应用程序，容器的选择通常会考虑应用程序所要处理的数据的位置，进行就近选择，从而实现“计算向数据靠拢”

•容器（Container）作为动态资源分配单位，每个容器中都封装了一定数量的CPU、内存、磁盘等资源，从而限定每个应用程序可以使用的资源量

•调度器被设计成是一个可插拔的组件，YARN不仅自身提供了许多种直接可用的调度器，也允许用户根据自己的需求重新设计调度器

•应用程序管理器（Applications Manager）负责系统中所有应用程序的管理工作，主要包括应用程序提交、与调度器协商资源以启动ApplicationMaster、监控ApplicationMaster运行状态并在失败时重新启动等

YARN体系结构

ApplicationMaster

ResourceManager接收用户提交的作业，按照作业的上下文信息以及从NodeManager收集来的容器状态信息，启动调度过程，为用户作业启动一个ApplicationMaster

ApplicationMaster的主要功能是：
（1）当用户作业提交时，ApplicationMaster与ResourceManager协商获取资源，ResourceManager会以容器的形式为ApplicationMaster分配资源；
（2）把获得的资源进一步分配给内部的各个任务（Map任务或Reduce任务），实现资源的“二次分配”；
（3）与NodeManager保持交互通信进行应用程序的启动、运行、监控和停止，监控申请到的资源的使用情况，对所有任务的执行进度和状态进行监控，并在任务发生失败时执行失败恢复（即重新申请资源重启任务）；
（4）定时向ResourceManager发送“心跳”消息，报告资源的使用情况和应用的进度信息；
（5）当作业完成时，ApplicationMaster向ResourceManager注销容器，执行周期完成。

YARN体系结构

NodeManager

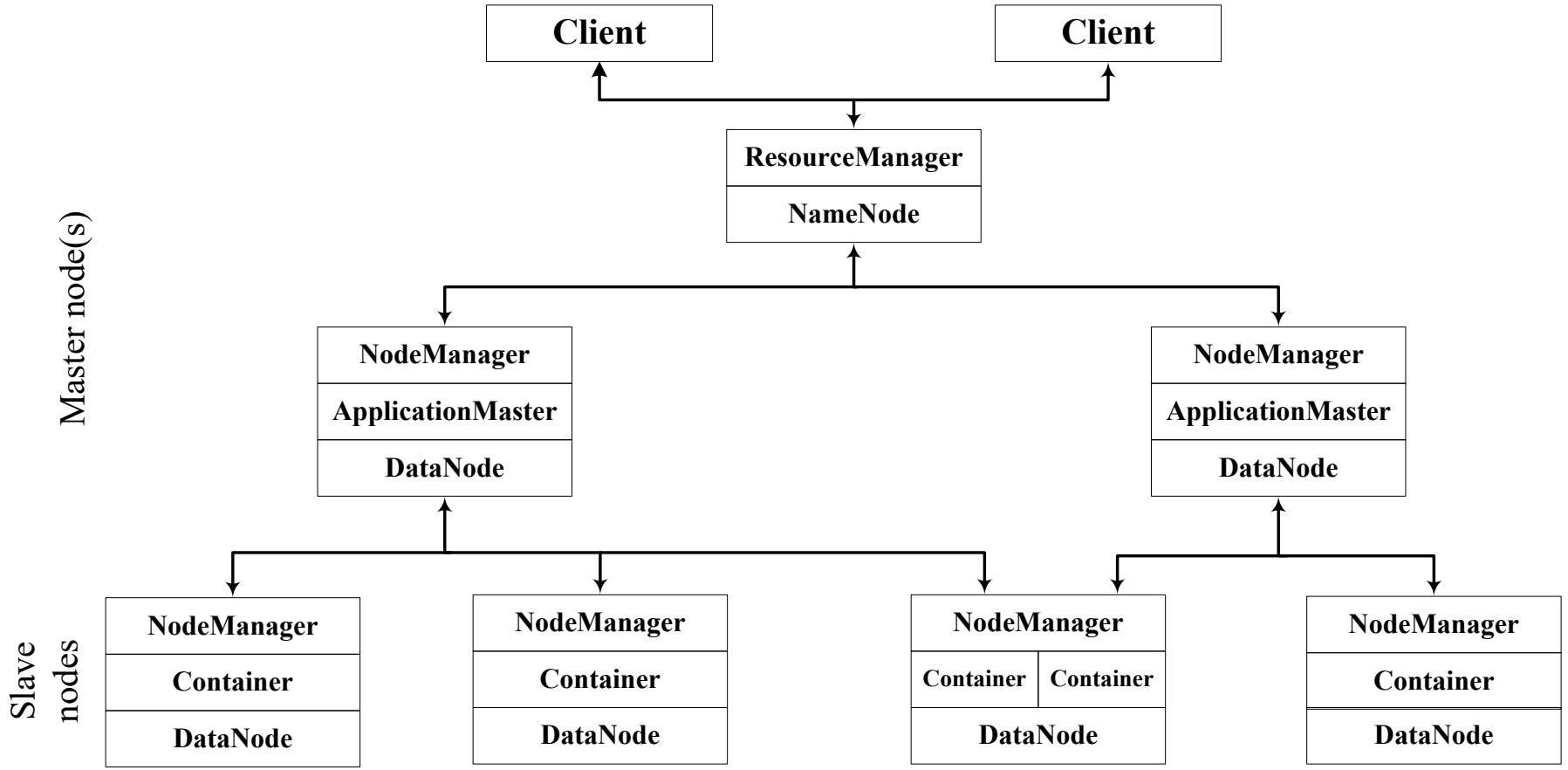
NodeManager是驻留在一个YARN集群中的每个节点上的代理，主要负责：

- 容器生命周期管理
- 监控每个容器的资源（CPU、内存等）使用情况
- 跟踪节点健康状况
- 以“心跳”的方式与ResourceManager保持通信
- 向ResourceManager汇报作业的资源使用情况和每个容器的运行状态
- 接收来自ApplicationMaster的启动/停止容器的各种请求

需要说明的是，NodeManager主要负责管理抽象的容器，只处理与容器相关的事情，而不具体负责每个任务（Map任务或Reduce任务）自身状态的管理，因为这些管理工作是由ApplicationMaster完成的，ApplicationMaster会通过不断与NodeManager通信来掌握各个任务的执行状态

YARN体系结构

在集群部署方面，YARN的各个组件是和Hadoop集群中的其他组件进行统一部署的



YARN和Hadoop平台其他组件的统一部署

YARN框架与MapReduce1.0框架的对比分析

•从MapReduce1.0框架发展到YARN框架，客户端并没有发生变化，其大部分调用API及接口都保持兼容，因此，原来针对Hadoop1.0开发的代码不用做大的改动，就可以直接放到Hadoop2.0平台上运行

总体而言，YARN相对于MapReduce1.0来说具有以下优势：

•大大减少了承担中心服务功能的ResourceManager的资源消耗
  •ApplicationMaster来完成需要大量资源消耗的任务调度和监控
  •多个作业对应多个ApplicationMaster，实现了监控分布化

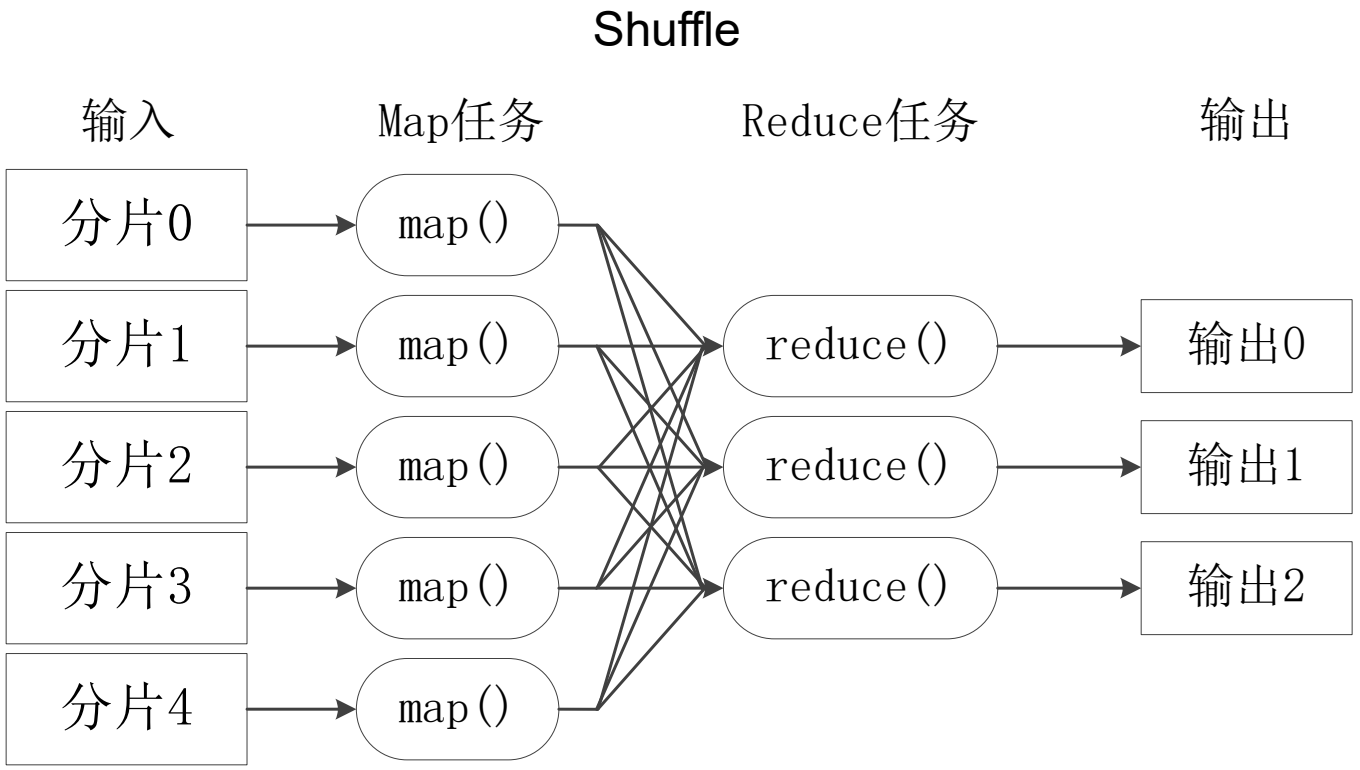
•MapReduce1.0既是一个计算框架，又是一个资源管理调度框架，但是，只能支持MapReduce编程模型。而YARN则是一个纯粹的资源调度管理框架，在它上面可以运行包括MapReduce在内的不同类型的计算框架，只要编程实现相应的ApplicationMaster

•YARN中的资源管理比MapReduce1.0更加高效
  •以容器为单位，而不是以slot为单位

MapReduce工作流程

- 工作流程概述
- MapReduce各个执行阶段
- Shuffle过程详解

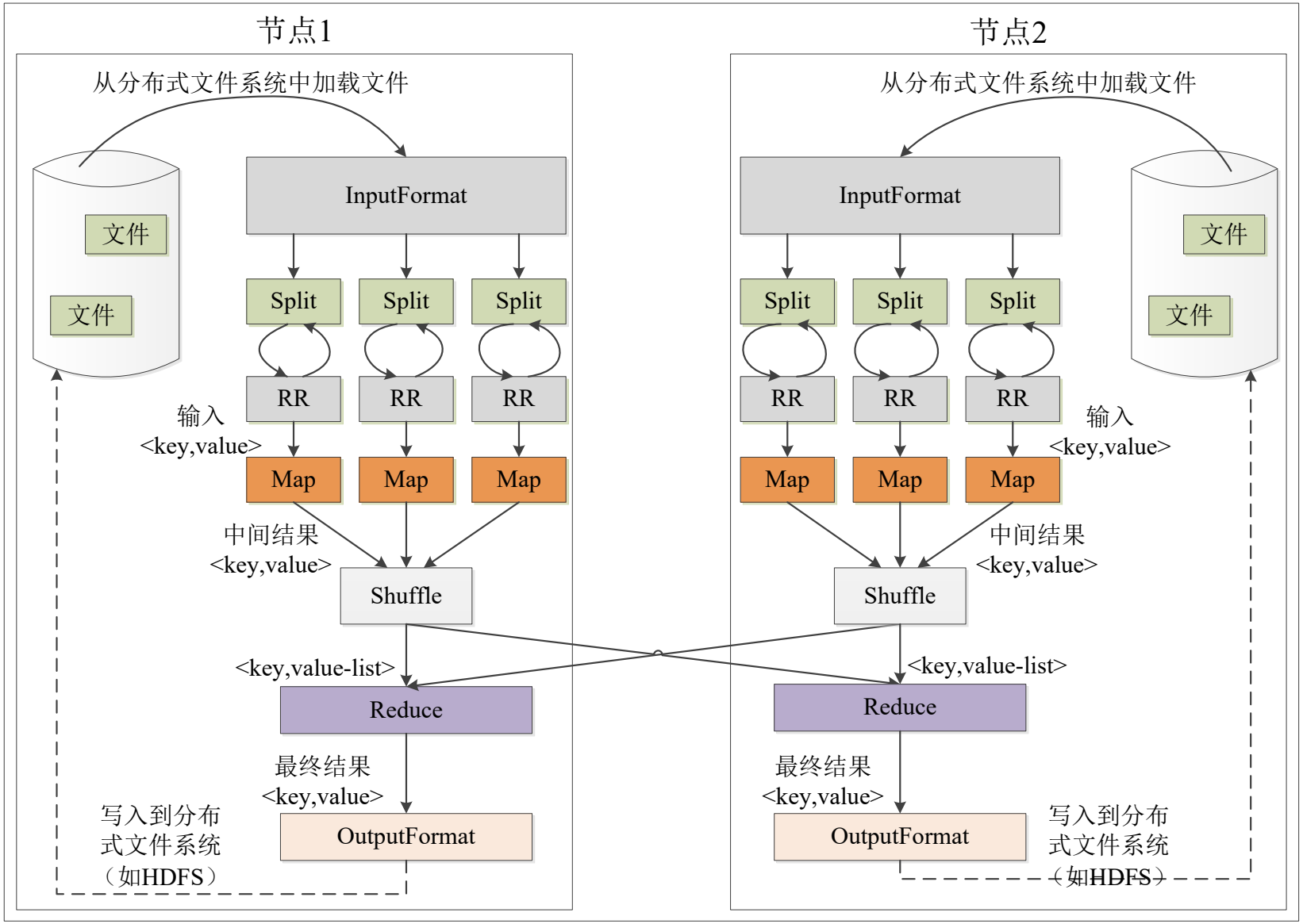
工作流程概述



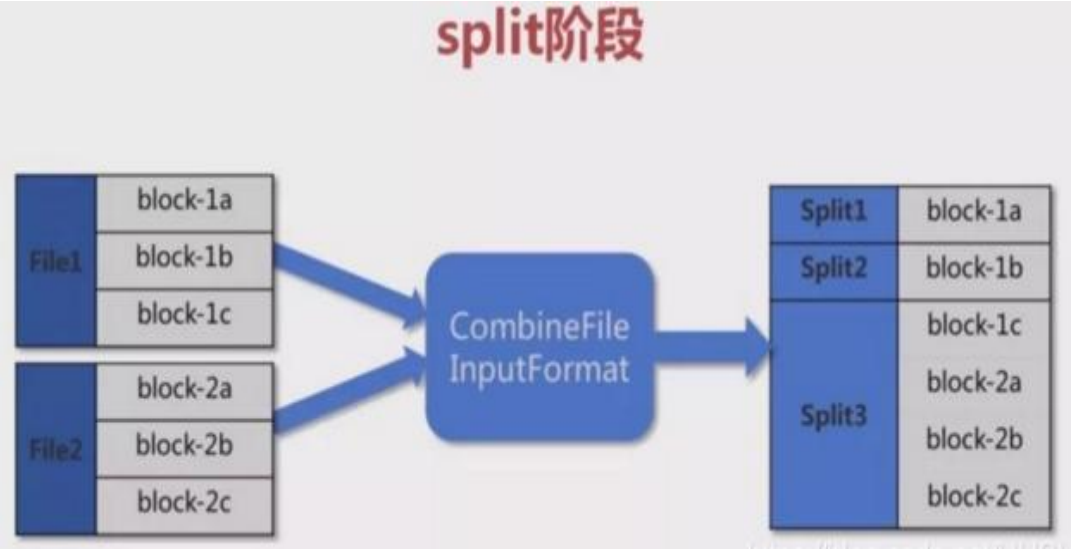
- 不同的Map任务之间不会进行通信
- 不同的Reduce任务之间也不会发生任何信息交换
- 用户不能显式地从一台机器向另一台机器发送消息
- 所有的数据交换都是通过MapReduce框架自身去实现的

（Key,Value）结构是数据IO关键

MapReduce各个执行阶段



Split & RR(RecordReader)



RR

1 , China is my motherland

2 , I love China

3 , ……..

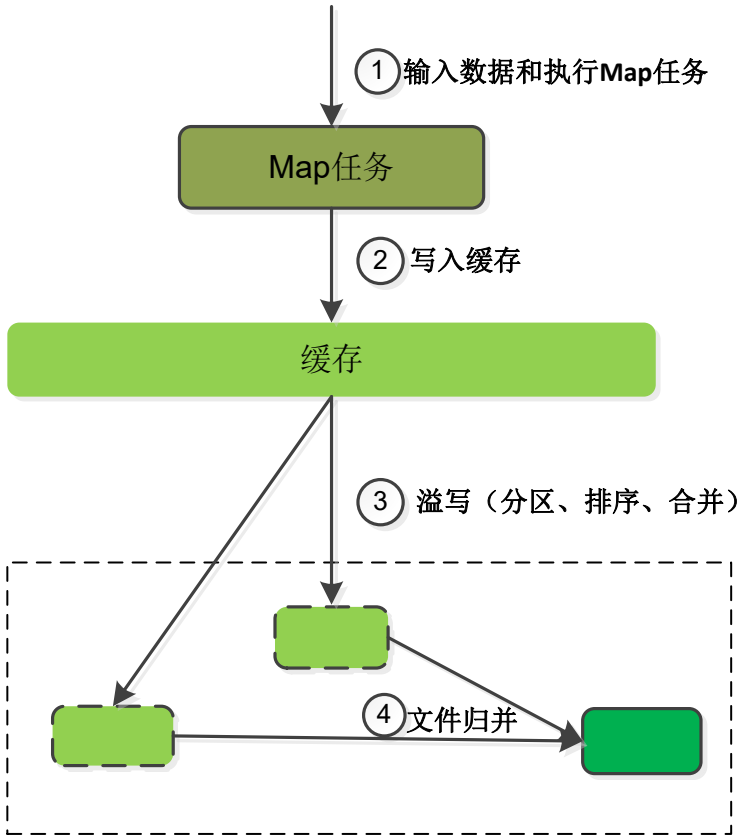
4 , …….

5 , ……..

6 , I am from China

Shuffle过程详解

Map端的Shuffle过程



- 每个Map任务分配一个缓存
- MapReduce默认100MB缓存

- 设置溢写比例0.8
- 分区默认采用哈希函数
- 排序是默认的操作
- 排序后可以合并（Combine）
- 合并不能改变最终结果

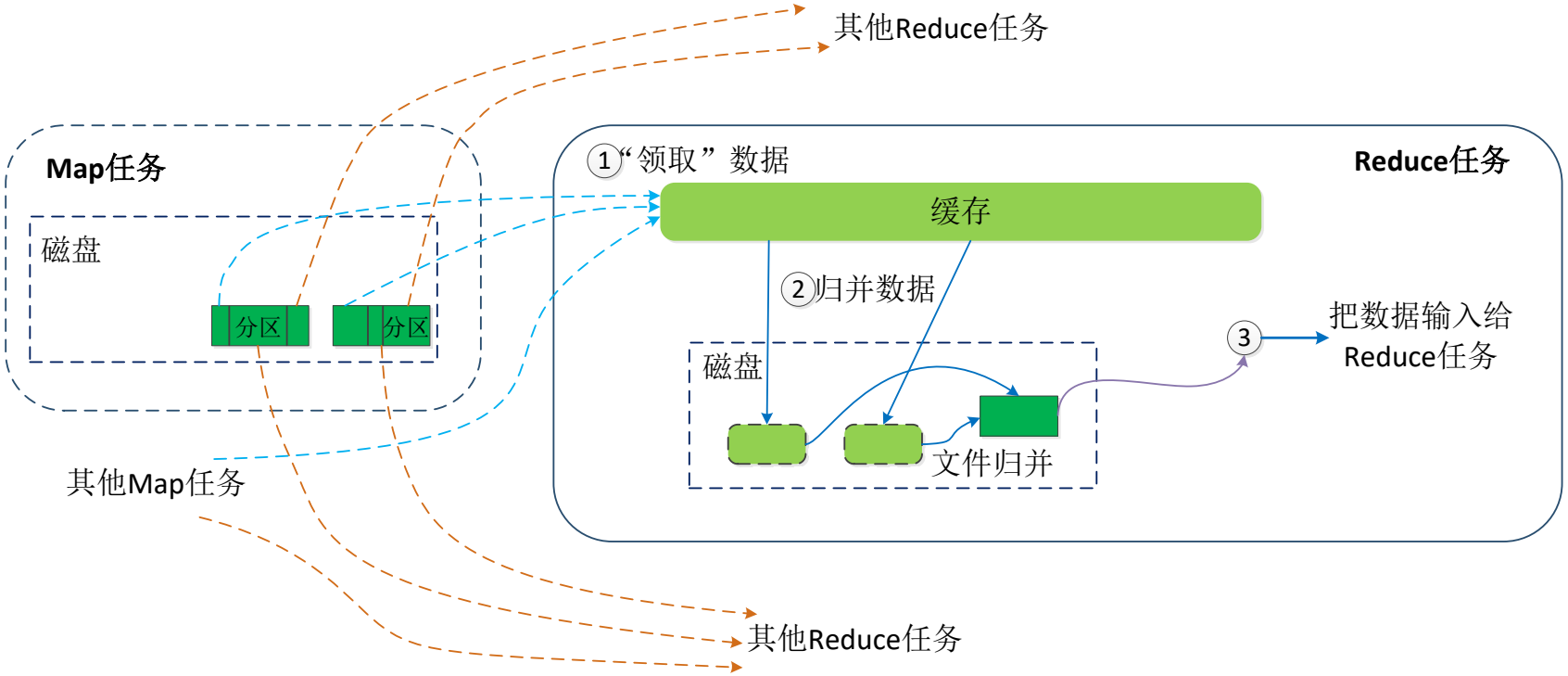
- 在Map任务全部结束之前进行归并
- 归并得到一个大的文件，放在本地磁盘
- JobTracker会一直监测Map任务的执行，并通知Reduce任务来领取数据

合并（Combine）和归并（Merge）的区别：
两个键值对<“a”,1>和<“a”,1>，如果合并，会得到<“a”,2>，如果归并，会得到<“a”,<1,1>>

Shuffle过程详解

Reduce端的Shuffle过程

- Reduce任务通过RPC向JobTracker询问Map任务是否已经完成，若完成，则领取数据
- Reduce领取数据先放入缓存，来自不同Map机器，先归并，再合并，写入磁盘
- 多个溢写文件归并成一个或多个大文件，文件中的键值对是排序的
- 当数据很少时，不需要溢写到磁盘，直接在缓存中归并，然后输出给Reduce

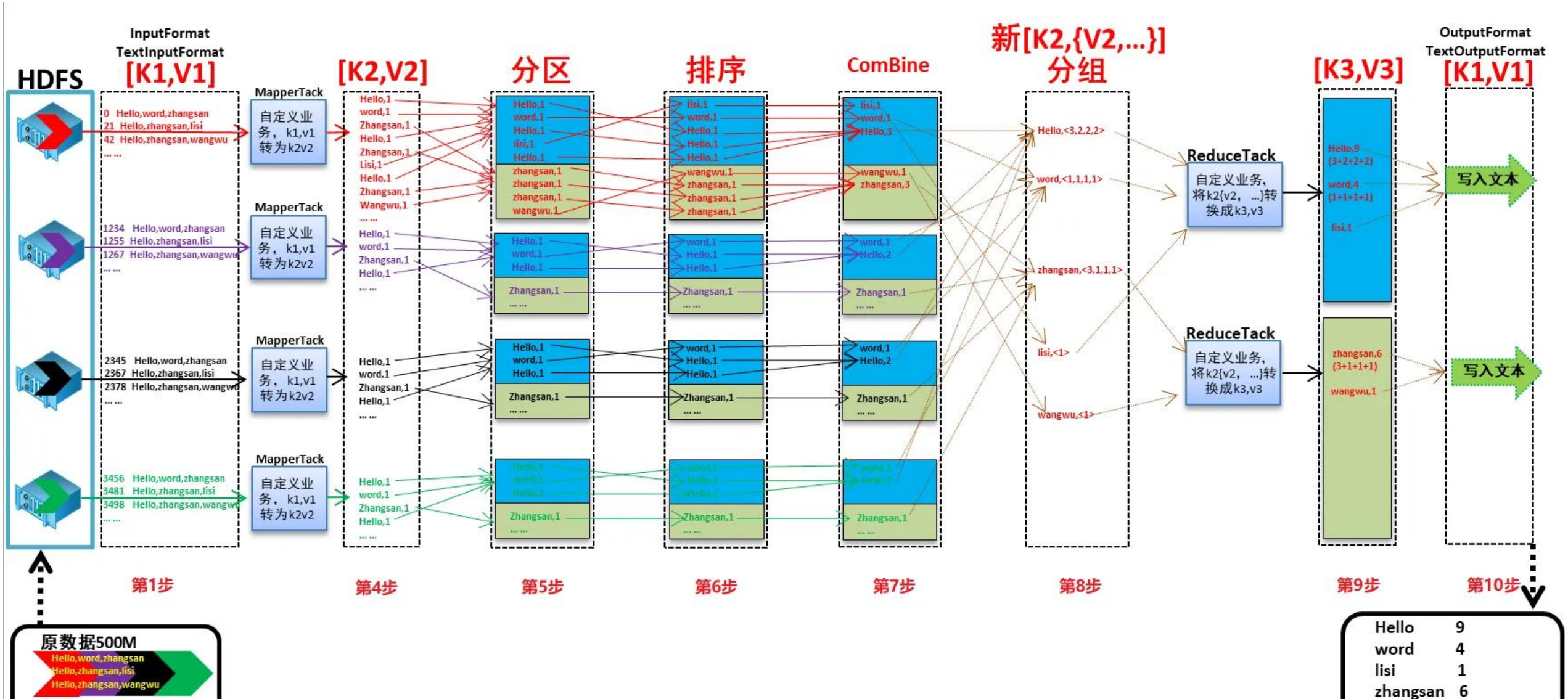


Reduce端的Shuffle过程

Mapreduce完整步骤

- 第1步：InputFormat InputFormat 到hdfs上读取数据-将数据传给Split
- 第2步：Split Split将数据进行逻辑切分，将数据传给RR
- 第3步：RR(RecordReader):将传入的数据转换成一行一行的数据，输出行首字母偏移量和偏移量对应的数据，将数据传给MAP
- 第4步：MAP :根据业务需求实现自定义代码，将数据传给Shuffle的partition
- 第5步：partition 分区:按照一定的分区规则，将key value的list进行分区。将数据传给Shuffle的Sort
- 第6步：Sort 排序:对分区内的数据进行排序，将数据传给Shuffle的combiner
- 第7步：combiner 合并:对数据进行局部聚合。将数据传给Shuffle的Group
- 第8步：Group|Merge 归并:将相同key的key提取出来作为唯一的key,将相同key对应的value获取出来作为value的list,将数据传给Reduce
- 第9步：Reduce ： 根据业务需求进行最终的合并汇总。将数据传给outputFormat
- 第10步：output 输出 :将数据写入HDFS

详细过程-（实例）


HDFS
InputFormat
TextInputFormat
[K1,V1]
MapperTack
自定义业务，k1,v1转为k2v2
[K2,V2]
分区
排序
ComBine
新[K2,{V2,...}]
分组
ReduceTack
自定义业务，将k2{v2，...}转换成k3,v3
[K3,V3]
OutputFormat
TextOutputFormat
[K1,V1]
写入文本
第1步
第4步
第5步
第6步
第7步
第8步
第9步
第10步
原数据500M
Hello,word,zhangsan
Hello,zhangsan,lisi
Hello,zhangsan,wangwu
Hello 9
word 4
lisi 1
zhangsan 6

实例分析：WordCount

- WordCount程序任务
- WordCount设计思路
- 一个WordCount执行过程的实例

WordCount程序任务

WordCount程序任务

程序	WordCount
输入	一个包含大量单词的文本文件
输出	文件中每个单词及其出现次数（频数），并按照单词字母顺序排序，每个单词和其频数占一行，单词和频数之间有间隔

一个WordCount的输入和输出实例

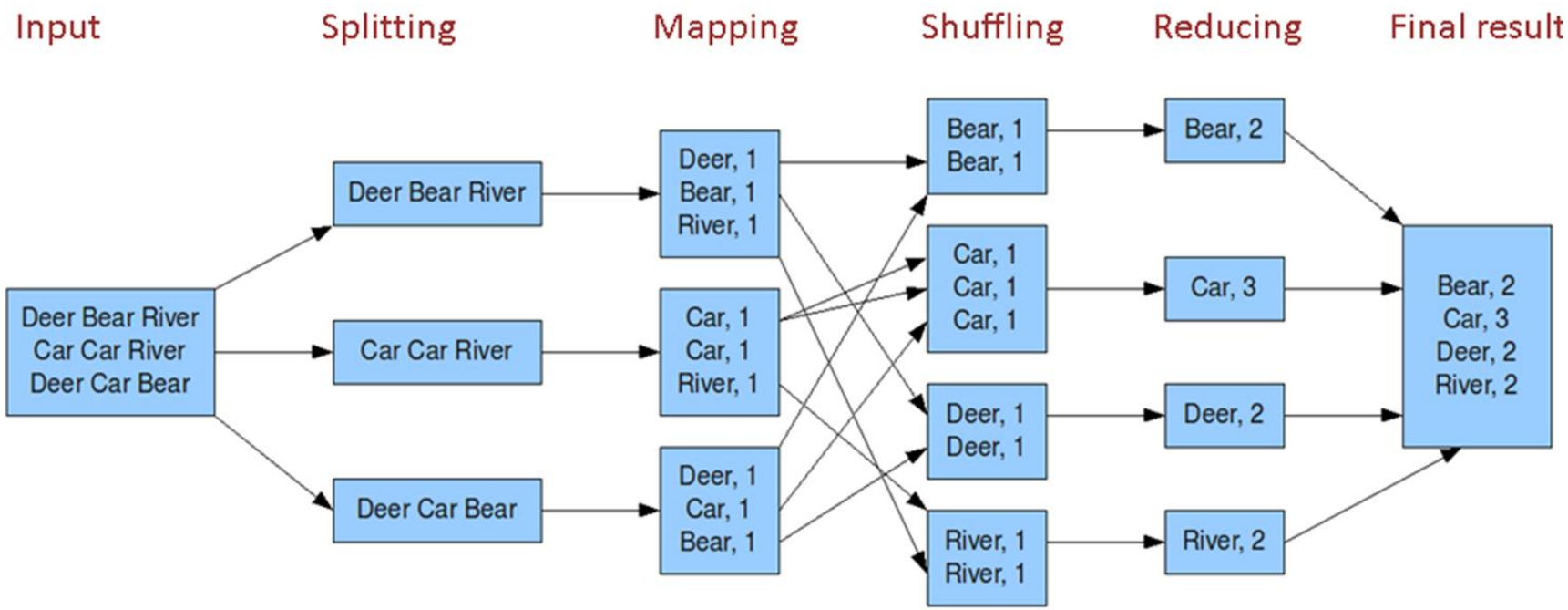
输入	输出
Hello World Hello Hadoop Hello MapReduce	Hadoop 1 Hello 3 MapReduce 1 World 1

WordCount设计思路

- 首先，需要检查WordCount程序任务是否可以采用MapReduce来实现
- 其次，确定MapReduce程序的设计思路
- 最后，确定MapReduce程序的执行过程

大多数情况下只需要设计Map和Reduce

WordCount mapreduce 过程



MapReduce　　缺点

- 性能慢—文件IO和网络传输
  - Spark
- 过于底层，Map和reduce两个函数的编写麻烦
  - 采用其他工具，如：对于数据库查询的Mapreduce操作用HIVE
- 并不是所有算法都能mapreduce，如很多机器学习算法

MapReduce编程实践

- 任务要求
- 编写Map处理逻辑
- 编写Reduce处理逻辑
- 编写main方法
- 编译打包代码以及运行程序
- Hadoop中执行MapReduce任务的几种方式

任务要求

文件A的内容如下：

China is my motherland

I love China

文件B的内容如下：

I am from China

期望结果如右侧所示：

I	2
is	1
China	3
my	1
love	1
am	1
from	1
motherland	1

编写Map处理逻辑

```
public class MyMapper extends
Mapper<Object,Text,Text,IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context) throws
IOException,InterruptedException{
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens())
        {
            word.set(itr.nextToken());
            context.write(word,one);
        }
    }
}
```

编写Reduce处理逻辑

```
package cn.edu.usst.mapreduce;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException,InterruptedException{
        int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
        result.set(sum);
        context.write(key,result);
    }
}
```

编写main方法

```
public class WordCount {
    public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();  //程序运行时参数
        String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();//获取命令行输
入参数
        if (otherArgs.length != 2)
        {       System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }
        Job job =Job.getInstance(conf,"word count");  //设置环境参数
        job.setJarByClass(WordCount.class);  //设置整个程序的类名
        job.setMapperClass(MyMapper.class);  //添加MyMapper类
        job.setReducerClass(MyReducer.class);  //添加MyReducer类
        job.setOutputKeyClass(Text.class);  //设置输出类型
        job.setOutputValueClass(IntWritable.class);  //设置输出类型
       // job.setCombinerClass(IntSumReducer.class);//开启combine
        FileInputFormat.addInputPath(job,new Path(otherArgs[0]));  //设置输入文件
        FileOutputFormat.setOutputPath(job,new Path(otherArgs[1])); //设置输出文件
        System.exit(job.waitForCompletion(true)?0:1);//提交任务到hadoop
    }
}
```

编译打包代码以及运行程序

实验步骤：

- 使用java编译程序
- 打包为jar包
- 上传JAR包到hadoop节点
- 运行jar包（需要启动Hadoop）
- 查看结果

Hadoop中执行MapReduce任务的几种方式

- Hadoop jar
- Pig
- Hive
- Python
- Shell脚本

执行过程

- 上传jar到任意hadoop节点
- 运行yarn和hdfs
- 上传数据文件到HDFS中
- 执行jar：
  - hadoop jar wordcount.jar 完整class名 数据文件所在目录路径 输出运算结果所在目录路径
  - 输出路径由hadoop自动创建
  - 路径均为HDFS

关于Mapreduce效率

- 编写容易，优化难
- 调优是关键