杰普软件科技有限公司

www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室

电话：（021）56778147

邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层

电话：（0512）50190290-8000

邮政编码：215311

电邮：training@briup.com

主页：http://www.briup.com

Briup High-End IT Training

Big Data

Brighten Your Way And Raise You Up.

代表什么意思？

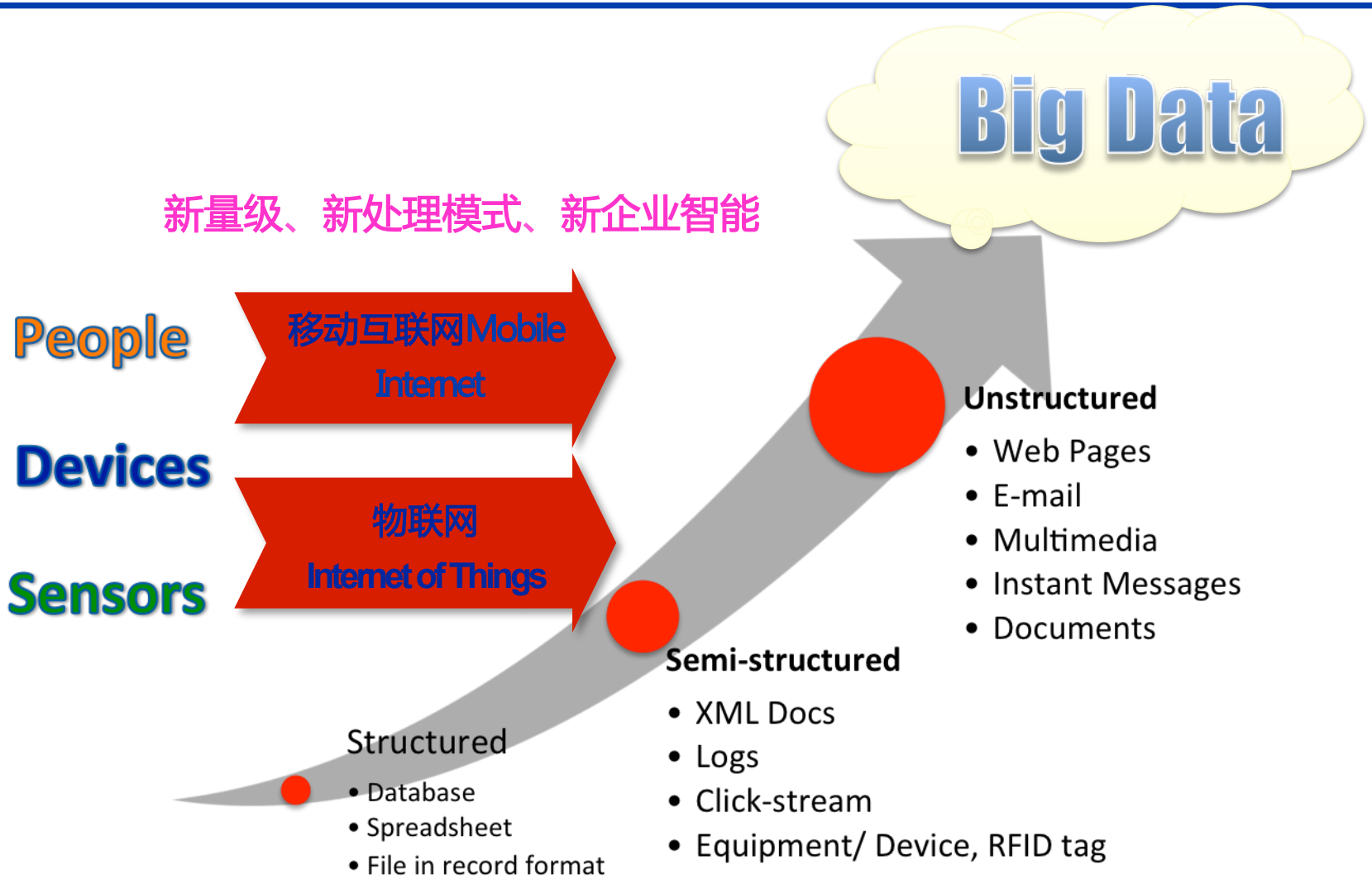
多多
5岁啦

引言

JD.京东 我的京东 返回京东首页 首页 账户设置 社区 消息 6

姓名	范范	钱丽	莫莫	张帆	刘勇	顾磊
无线鼠标	√			√	√	
Linux编程思想	√		√	√		
面膜		√				
乐高		√				√
情趣用品	√			√		
皮鞋	√	√		√		
小米手环	√		√			√

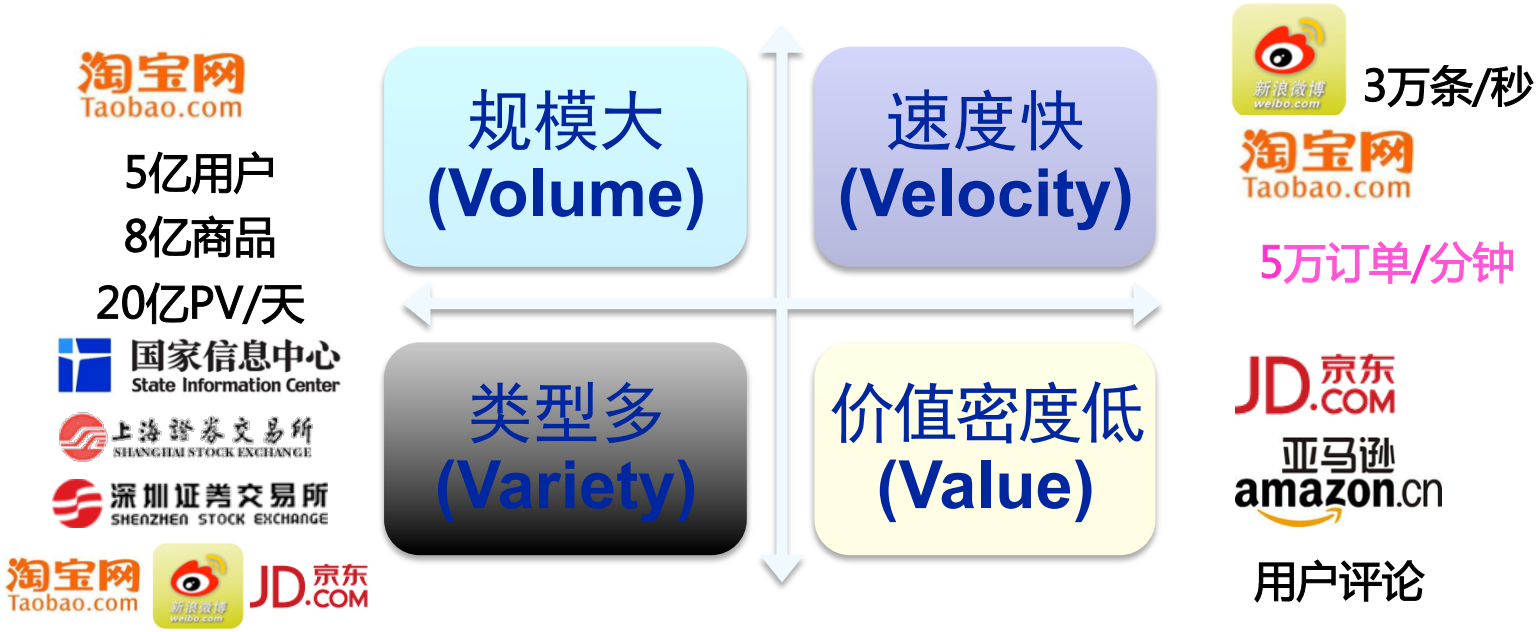
大数据(Big Data)时代来临



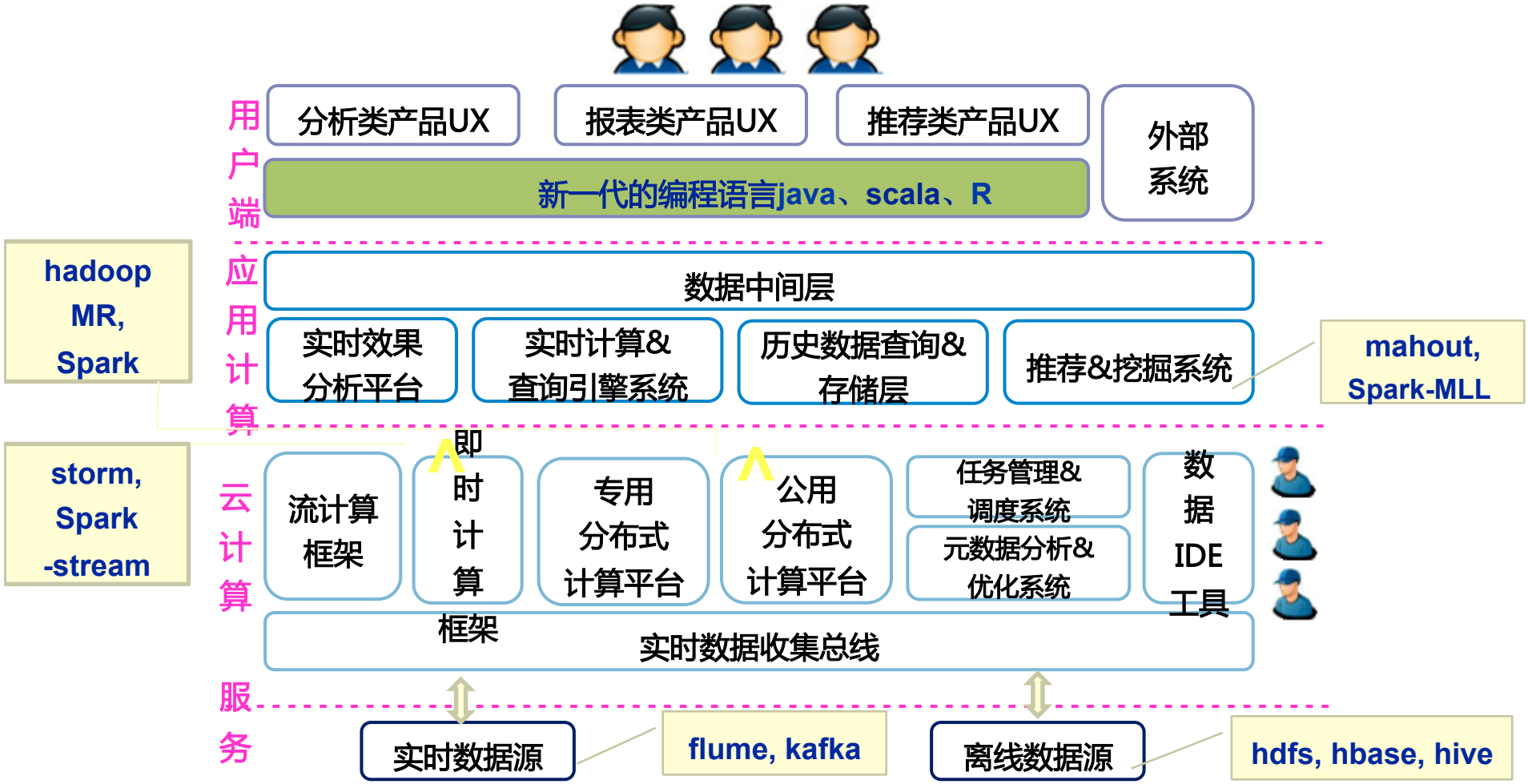
大数据表象概念：百度数据规模

大数据定义及特点

大数据是通过传统数据库技术和数据处理工具不能处理的庞大而复杂的数据集合。



大数据的系统架构：整体系统架构



几个概念

集群
cluster

数据密集型
DIC

计算密集型
CIC

向上扩
展Scale-Up

向外扩
展Scale-Out

机器学
习Machine
Learning

云计算
Cloud
Computing

Tel： 021-56657112 Fax： 021-55661523-8003

8

- 认识Hadoop及其生态圈
- 构建Hadoop集群
- HDFS操作与编程
- MapReduce程序设计及优化
- MapReduce高级应用及程序运行机制
- Yarn管理框架
- Sqoop2使用
- Zookeeper使用与原理

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

初识Hadoop

Brighten Your Way And Raise You Up.

大数据的时代

- 在Web2.0时代，人们从信息的被动接收者变成了主动创造者
  - 互联网：社交网络、视频、图片、电子商务
  - 物联网：移动设备、传感器
  - 天文、地理、环境、气象、交通信息
  - 扫描书籍、历史文献、社会交互信息
  - 医疗扫描、电子病历

如何面对海量数据？

- 古代，人们用牛来拉重物。当一头牛拉不动一根圆木时，他们不曾想过培育更大更壮的牛。同样，我们也不需要尝试开发超级计算机，而应试着结合使用更多计算机系统。

于是……Hadoop应运而生！

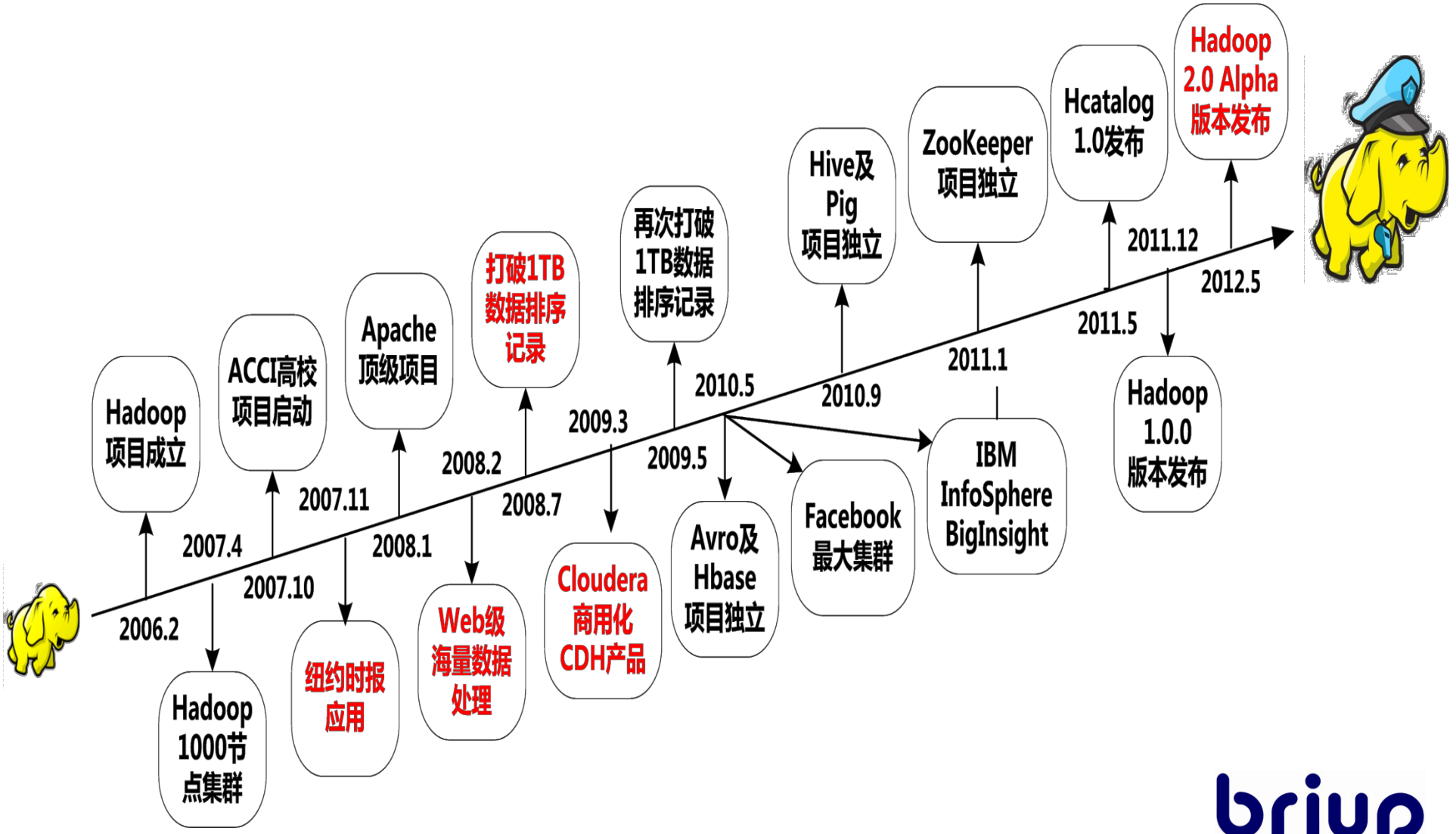
Hadoop的优势

- 方便------->不需昂贵和高可靠的硬件资源，商用硬件即可
- 弹性------->集群节点很容易扩展或卸载
- 健壮------->故障检测和自动恢复
- 简单------->用户可以快速编写出高效的并行分布代码

Hadoop之父Doug Cutting

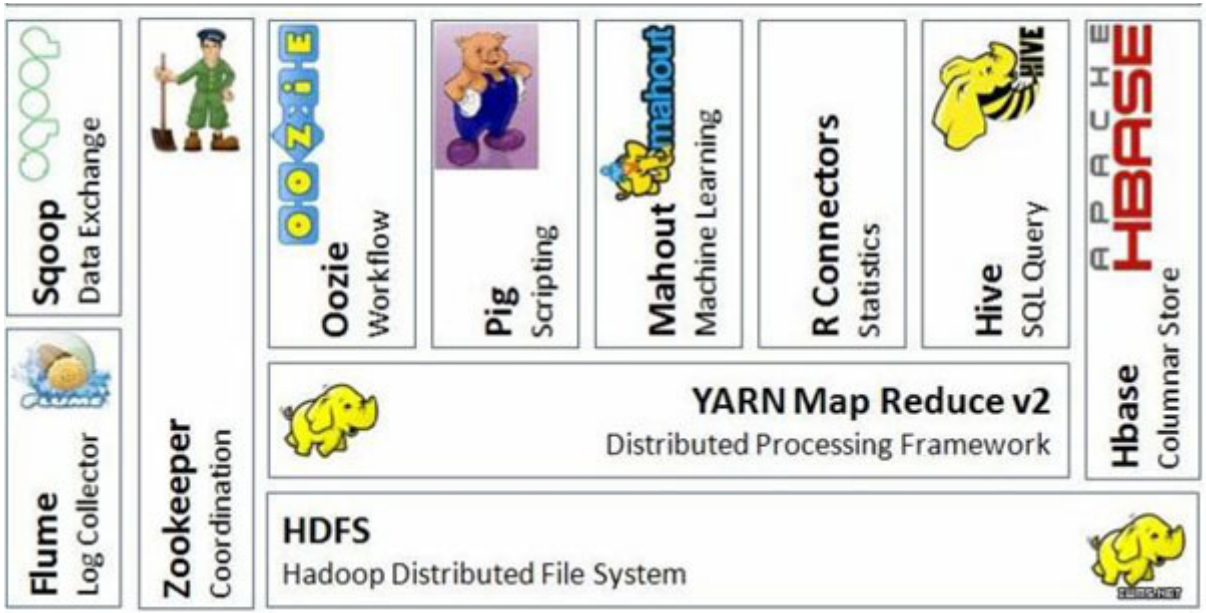
- 相关项目
  - Apache Lucene -------> Nutch ------->Hadoop
  - 文本搜索系统库------>网络搜索引擎------->海量数据存储和处理
- 灵感来源
  - GFS -------> HDFS
  - MapReduce -------> MapReduce
  - BigTable -------> Hbase
- 成功案例
  - Yahoo!
  - 纽约时报（4TB报纸/100台计算机/历时不到24小时）
- 爆发式发展
  - 大量的相关项目：Hive、Hbase、ZooKeeper、sqoop…..
  - 专注于Hadoop的公司：Cloudera　　Hortonworks ….
- Hadoop生态圈形成

Hadoop的发展历程



Hadoop生态圈

- 经过几年的发展，Hadoop已发展成包含多个相关项目的软件生态系统
  - 狭义的Hadoop:
    - 核心项目-->Common、 HDFS、 MapReduce
  - 广义的Hadoop:
    - 核心项目+其他项目-->Avro、ZooKeeper、Hive、Pig、Hbase等
    - 以上述为基础，面向具体领域或应用的项目------->Mahout、X-Rime、Crossbow、Ivory……
    - 数据交换、工作流等外围支撑系统------->Chukwa、Flume、Sqoop、Oozie……



为Hadoop其他项目提供一些常用工具，如系统配置工具Configuration、远程过程调用RPC、序列化机制、Hadoop抽象文件系统FileSystem等

分布式文件系统，运行于大型商用机集群，是Hadoop体系中海量数据存储管理的基础

分布式数据处理模型和执行环境，是Hadoop体系中海量数据处理的基础

- 一个分布式的面向列的数据库
- 构建在HDFS之上。
- 适用于Hadoop应用需要实时读/写随机访问非常大型数据集

- 一个分布式的服务框架，解决了分布式计算中的一致性问题。
- 可用于解决分布式应用中遇到的数据管理问题，如统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等
- 常作为其他Hadoop相关项目的主要组件。

一种数据流语言和运行环境，用以检索非常大的数据集。Pig本身运行在HDFS和MapReduce集群上。

- 最早由Facebook设计，建立在Hadoop基础上的数据仓库框架。
- Hive管理HDFS中存储 的数据，并提供基于SQL查询语言以查询数据。

一个高可用的，高可靠的，分布式的海量数据采集、聚合和传输的系统，经常用做日志采集器

Apache sqoop

- SQL-to-Hadoop的缩写，主要作用是在结构化数据存储和Hadoop之间进行数据转换。
- 是一种在数据库和HDFS之间高效传输数据的工具

- 一个机器学习和数据挖掘的库，提供用于聚类、回归测试和统计建模常见算法的MapReduce实现

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop客户端

Brighten Your Way And Raise You Up.

1. 前期准备

- Linux
- JDK

2. 安装Hadoop

- 解压下载好的hadoop-x.x.x.tar.gz
  - tar -zvxf hadoop-2.6.0.tar.gz –C /opt
- 编辑hadoop-2.6.0/etc/hadoop/hadoop-env.sh
  - export JAVA_HOME=/opt/jdk1.7.0_79
- 在.bashrc下配置PATH
  - PATH=$PATH:/opt/hadoop-2.6.0/bin:/opt/hadoop-2.6.0/sbin
- 验证是否安装成功
  - hadoop version

1. 客户端配置之core-site.xml

- **core-site.xml**

```
<configuration>
        <!-- 指定Hadoop所使用的文件系统Schema-->
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://172.16.0.101:9000</value>
        </property>
</configuration>
```

2. 客户端配置之mapred-site.xml

- **mapred-site.xml**

```
<configuration>
        <!-- 指定MapReduce程序运行在Yarn上-->
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

3. 客户端配置之yarn-site.xml

- yarn-site.xml

```
<configuration>
        <!-- 指定ResourceManager的地址-->
        <property>
                <name>yarn.resourcemanager.hostname</name>
                <value>172.16.0.101</value>
        </property>
</configuration>
```

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop分布式环境搭建

Brighten Your Way And Raise You Up.

1. 前期准备

- Linux
- JDK

2. 安装Hadoop

- 解压下载好的hadoop-x.x.x.tar.gz
  - tar -zvxf hadoop-2.6.0.tar.gz
- 编辑hadoop-2.6.0/etc/hadoop/hadoop-env.sh
  - export JAVA_HOME=/opt/jdk1.7.0_79
- 在.bashrc下配置PATH
  - PATH=...../home/briup/hadoop-2.6.0/bin:/home/briup/hadoop-2.6.0/sbin
- 验证是否安装成功
  - hadoop version

Hadoop的三种启动模式

- 独立模式(Local或Standalone Mode)
  - 默认情况下，hadoop即处于该模式，用于开发和调试
- 伪分布模(Pseudo-Distributed Mode)
  - Hadoop的守护进程运行在本机机器，模拟一个小规模的集群
- 全分布模式(Fully-Distributed Mode)
  - Hadoop的守护进程运行在一个集群上

Hadoop的配置

- Hadoop的各个组件均可利用XML进行配置，这些文件存放于etc/hadoop目录
  - Common组件 ------> core-site.xml
  - HDFS组件 ------> hdfs-site.xml
  - MapReduce组件 ------> mapred-site.xml
  - YARN组件 ------>yarn-site.xml

3. 分布模式配置之core-site.xml

- core-site.xml

```
<configuration>
        <!-- 指定Hadoop所使用的文件系统Schema-->
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://ip:9000</value>
        </property>
</configuration>
```

4. 分布模式配置之hdfs-site.xml

- hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.nameservices</name>
                <value>hadoop-cluster1</value>
        </property>
        <property>
                <name>dfs.replication</name>
                <value>1</value>
        </property>
        <property>
                <name>dfs.namenode.secondary.http-address</name>
                <value>ip:50090</value>
         </property>
```

4. 分布模式配置之hdfs-site.xml

- hdfs-site.xml

```
<property>
        <name>dfs.namenode.name.dir</name>
        <value>file:/data/hadoop/hdfs/nn</value>
</property>
<property>
        <name>fs.checkpoint.dir</name>
        <value>file:/data/hadoop/hdfs/snn</value>
</property>
<property>
        <name>fs.checkpoint.edits.dir</name>
        <value>file:/data/hadoop/hdfs/snn</value>
</property>
<property>
        <name>dfs.datanode.data.dir</name>
        <value>file:/data/hadoop/hdfs/dn</value>
</property>
</configuration>
```

5. 分布模式配置之mapred-site.xml

- mapred-site.xml

```
<configuration>
        <!-- 指定MapReduce程序运行在Yarn上-->
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

6. 分布模式配置之yarn-site.xml

- yarn-site.xml

```
<configuration>
        <!-- 指定ResourceManager的地址-->
        <property>
        <name>yarn.resourcemanager.hostname</name>
                <value>ip</value>
        </property>
        <!-- 指定reducer获取数据的方式-->
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.nodemanager.local-dirs</name>
                <value>/data/hadoop/yarn/nm</value>
        </property>
</configuration>
```

7. 配置其他服务器

- 将上述配置文件拷贝到其他服务器对应目录下
- 在相应的服务器上执行如下命令启动服务
  - 启动名字节点：**hadoop-daemon.sh start namenode**
  - 启动第二名字节点：**hadoop-daemon.sh start secondarynamenode**
  - 启动数据节点：**hadoop-daemon.sh start datanode**
  - 启动资源管理器：**yarn-daemon.sh start resourcemanager**
  - 启动节点管理器：**yarn-daemon.sh start nodemanager**

8. 格式化HDFS文件系统

- hdfs namenode –format

9. 配置SSH无密码登录

- 测试
  - ssh localhost
- 测试未通过
  - 切换目录
    - cd ~/.ssh
  - 生成公钥（id_dsa.pub）和私钥文件(id_dsa)
    - ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
  - 将公钥拷贝到要免登陆的机器上(以下二选一)
    - cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys
    - ssh-copy-id -i localhost

6. 启动HDFS和MapReduce守护进程

- start-dfs.sh
  - 监控平台：
    - ✓ http://IP:50070

7. 启动MapReduce守护进程

- start-yarn.sh
  - 监控平台：
    - http://IP:8088
- mr-jobhistory-daemon.sh start historyserver
  - 监控平台
    - http://IP:19888

8. 关闭守护进程

- 终止HDFS守护进程
  - stop-dfs.sh
- 终止YARN守护进程
  - stop-yarn.sh
  - mr-jobhistory-daemon.sh start historyserver

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop之HDFS

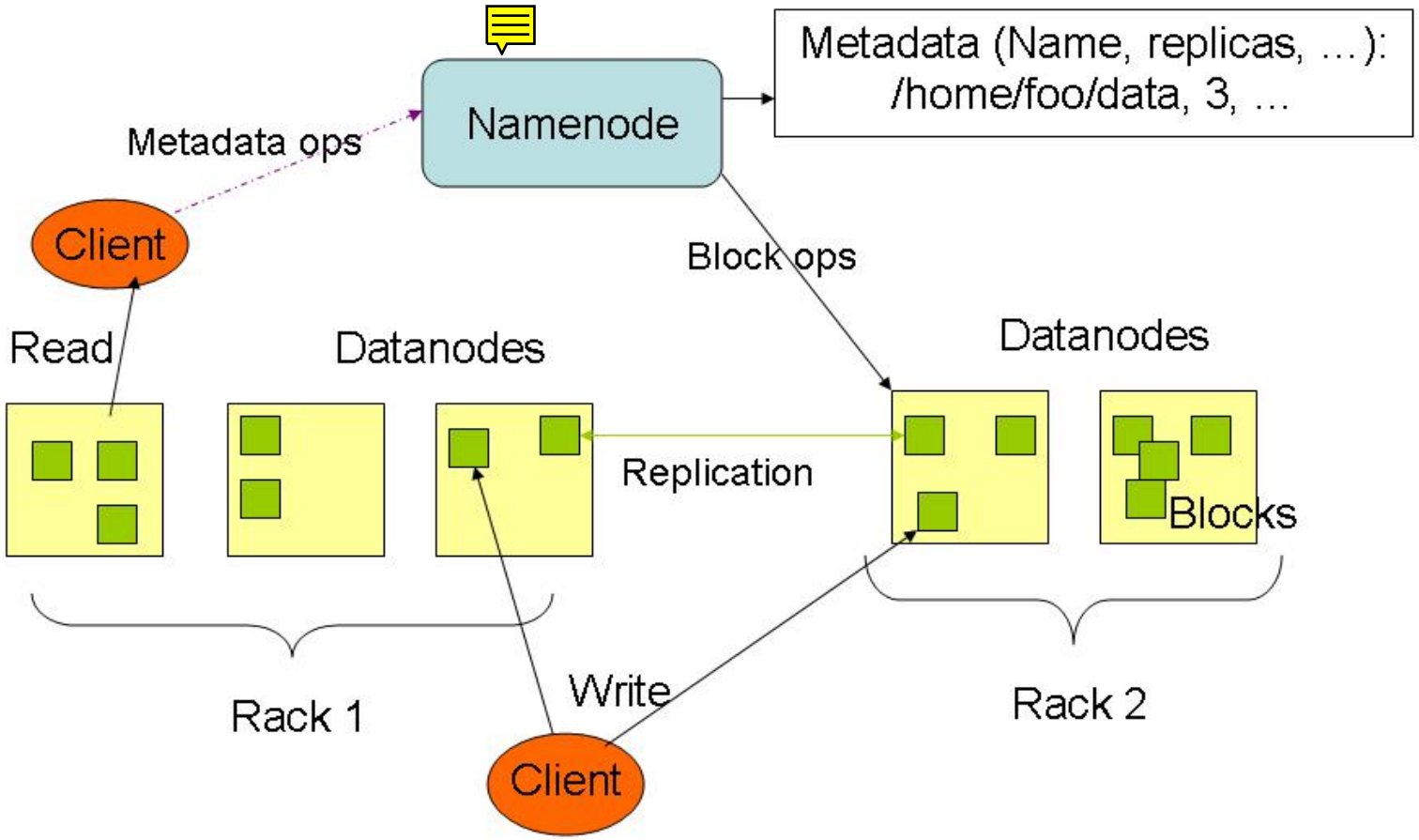
Brighten Your Way And Raise You Up.

分布式文件系统

- 问题概述
  - 海量数据超过单台物理计算机的存储能力
- 解决方案
  - 对数据分区存储于若干物理主机
- 分布式文件系统应运而生
  - 管理网络中跨多台计算机存储的文件系统
  - HDFS就是这样的一个分布式文件系统

- 超大文件(TB级或PB级)
- 流式数据访问，“一次写入，多次读取”的高效访问模式
- 商用硬件，不需要昂贵而高可靠的硬件
- 不适合低时间延迟的数据访问
- 不适宜大量的小文件
- 主从架构

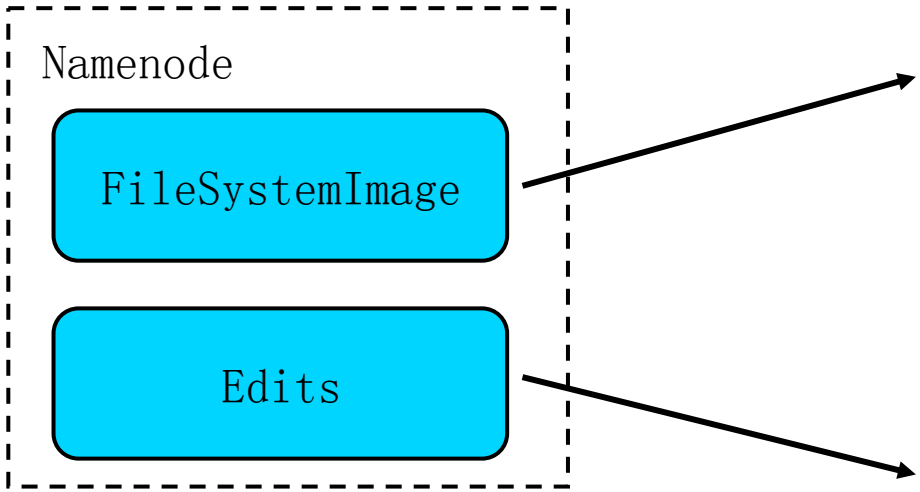
HDFS主从结构体系

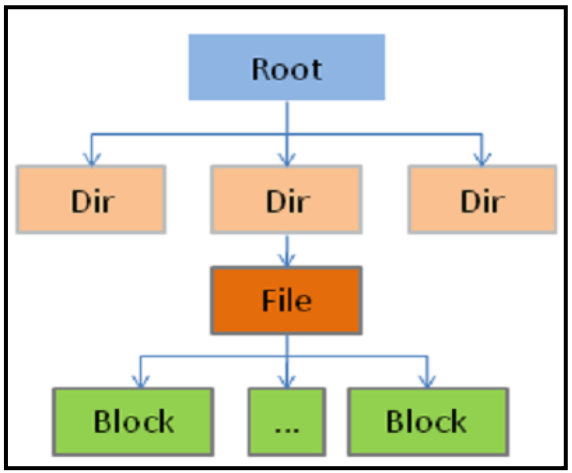

Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

HDFS基本概念--NameNode

- HDFS集群中两大类节点中的主节点
- 维护着文件系统树中所有文件和目录的元数据
- 协调客户端对文件的访问

命名空间树缓存储在RAM，并持久化
在FSImage本地文件中





记录创建、删除等操作

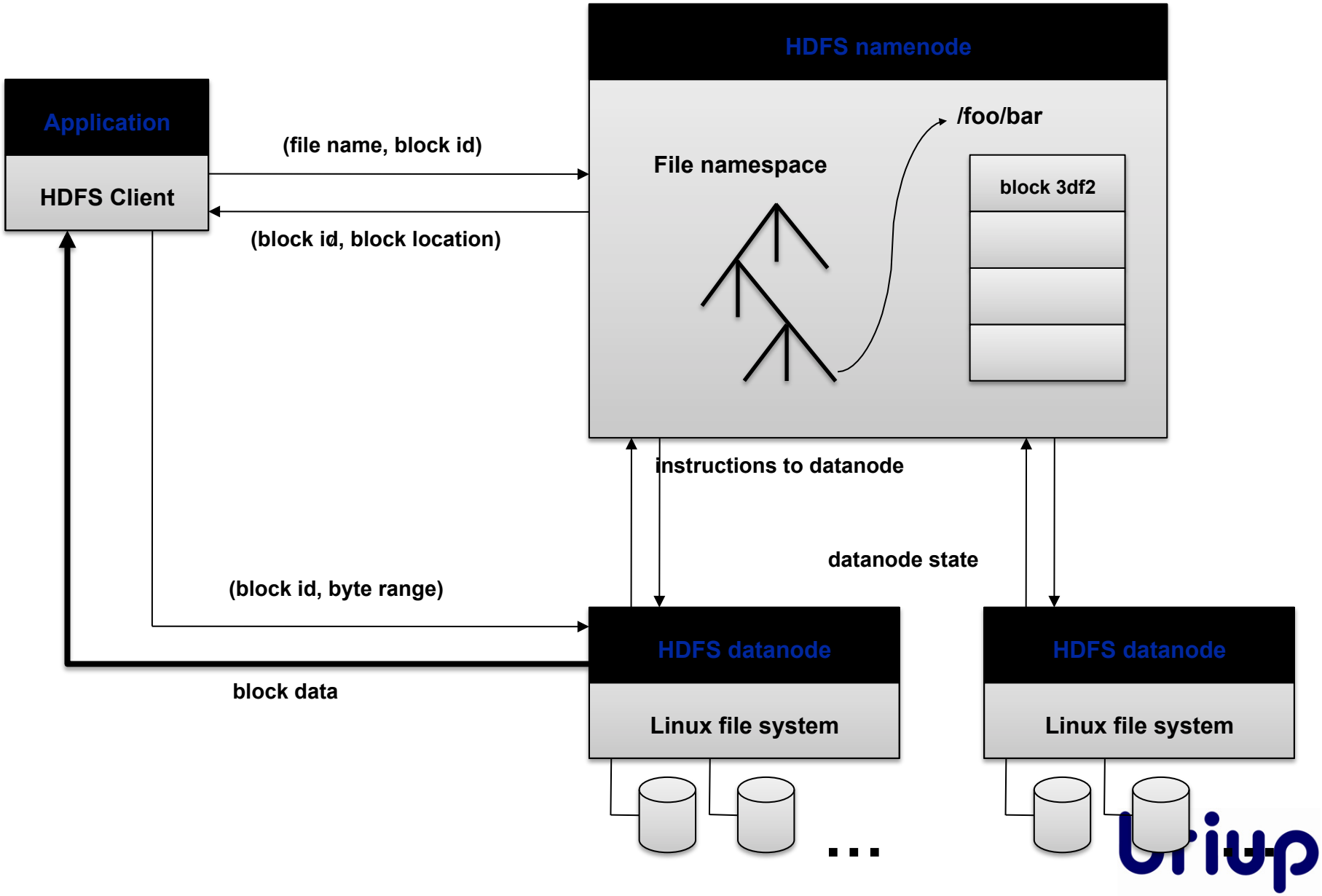
Edits日志记录打开、关闭等操作文件
或目录等操作

HDFS基本概念--DataNode

- HDFS集群中两大类节点的从节点
- 接收客户端或NameNode的调度
- 储存和检索数据块
- 定期向NameNode发送自身存储的数据块列表

HDFS基本概念--Block

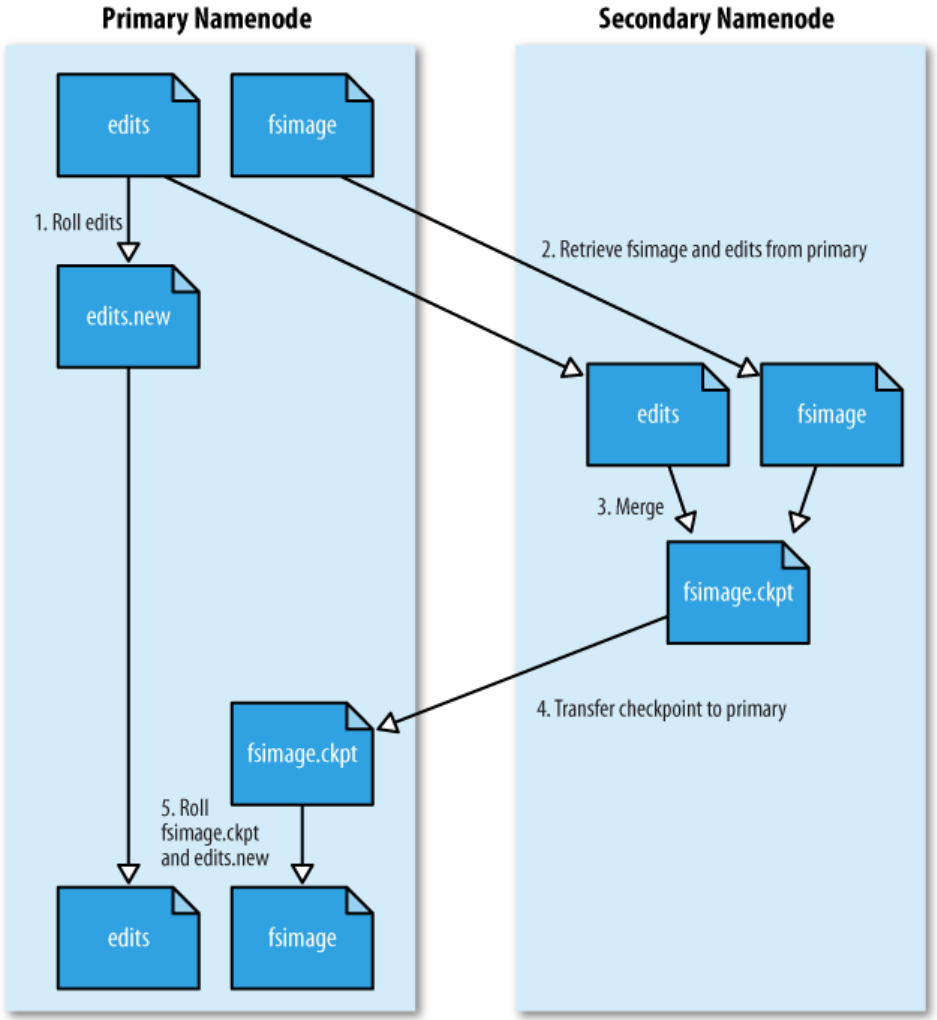
- HDFS中的最小存储单元


HDFS namenode
Application
HDFS Client
(file name, block id)
(block id, block location)
File namespace
/foo/bar
block 3df2
instructions to datanode
datanode state
(block id, byte range)
block data
HDFS datanode
Linux file system
HDFS datanode
Linux file system
...

保障可靠性的措施

- 冗余备份
  - 每个文件存储成一系列数据块（Block）。为了容错，文件的所有数据块都会有副本（副本数量即复制因子，可配置）
- 副本存放
  - 采用机架感知（Rack-aware）的策略来改进数据的可靠性、可用性和网络带宽的利用率
- 心跳检测
  - NameNode周期性地从集群中的每个DataNode接受心跳包和块报告，收到心跳包说明该DataNode工作正常
- 安全模式
  - 系统启动时，NameNode会进入一个安全模式。此时不会出现数据块的写操作
- 数据完整性检测
  - HDFS客户端软件实现了对HDFS文件内容的校验和（Checksum）检查

HDFS之SecondaryNamenode



- 定期合并edits和fsImage
- 防止edits日志文件过大
- 提供NameNode的fsImage文件的检查点，以实现NameNode故障恢复

HDFS高可用性(HA)

- 单点失效问题
  - 如果NameNode失效，那么客户端或MapReduce作业均将无法读写查看文件
- 解决方案
  - 启动一个拥有文件系统元数据的新NameNode(时间长)
  - 配置一对活动-备用(Active-Standby)NameNode,活动NameNode失效时，备用NameNode立即接管，用户不会有明显中断感觉
    - ✓ 共享编辑日志文件（借助NFS、zookeeper等）
    - ✓ DataNode同时向两个NameNode汇报数据块信息
    - ✓ 客户端采用特定机制处理NameNode失效问题，该机制对用户透明
- Hadoop2.X提供了对HA的支持

HDFS联邦(Federation)

- 横向扩展瓶颈
  - NameNode内存限定了拥有大量文件的超大集群的横向扩展
- Hadoop2.X解决方案
  - 运行系统添加namenode实现扩展
  - 每个namenode管理文件命名系统的空间的一部分

- 查看命令帮助------>hdfs -help

```
hdfs@host1:~$ hdfs -help
Usage: hdfs [--config confdir] COMMAND
       where COMMAND is one of:
  dfs                  run a filesystem command on the file systems supported in Hadoop.
  namenode -format     format the DFS filesystem
  secondarynamenode    run the DFS secondary namenode
  namenode             run the DFS namenode
  journalnode          run the DFS journalnode
  zkfc                 run the ZK Failover Controller daemon
  datanode             run a DFS datanode
  dfsadmin             run a DFS admin client
  haadmin              run a DFS HA admin client
  fsck                 run a DFS filesystem checking utility
  balancer             run a cluster balancing utility
  jmxget               get JMX exported values from NameNode or DataNode.
  mover                run a utility to move block replicas across
                       storage types
  oiv                  apply the offline fsimage viewer to an fsimage
  oiv_legacy           apply the offline fsimage viewer to an legacy fsimage
  oev                  apply the offline edits viewer to an edits file
  fetchdt              fetch a delegation token from the NameNode
  getconf              get config values from configuration
  groups               get the groups which users belong to
  snapshotDiff         diff two snapshots of a directory or diff the
                       current directory contents with a snapshot
  lsSnapshottableDir   list all snapshottable dirs owned by the current user
                                                 Use -help to see options
  portmap              run a portmap service
  nfs3                 run an NFS version 3 gateway
  cacheadmin           configure the HDFS cache
  crypto               configure HDFS encryption zones
  storagepolicies      get all the existing block storage policies
  version              print the version
```

- 运行一个文件系统命令在Hadoop集群支持的文件系统上

```
hdfs@host1:~$ hdfs dfs -help
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] [-l] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-count [-q] [-h] <path> ...]
        [-cp [-f] [-p | -p[topax]] <src> ... <dst>]
        [-createSnapshot <snapshotDir> [<snapshotName>]]
        [-deleteSnapshot <snapshotDir> <snapshotName>]
        [-df [-h] [<path> ...]]
        [-du [-s] [-h] <path> ...]
        [-expunge]
        [-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
        [-getfacl [-R] <path>]
        [-getfattr [-R] {-n name | -d} [-e en] <path>]
        [-getmerge [-nl] <src> <localdst>]
        [-help [cmd ...]]
        [-ls [-d] [-h] [-R] [<path> ...]]
        [-mkdir [-p] <path> ...]
        [-moveFromLocal <localsrc> ... <dst>]
        [-moveToLocal <src> <localdst>]
        [-mv <src> ... <dst>]
        [-put [-f] [-p] [-l] <localsrc> ... <dst>]
        [-renameSnapshot <snapshotDir> <oldName> <newName>]
        [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
        [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
        [-setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>]|[--set <acl_spec> <path>]]
        [-setfattr {-n name [-v value] | -x name} <path>]
        [-setrep [-R] [-w] <rep> <path> ...]
        [-stat [format] <path> ...]
        [-tail [-f] <file>]
        [-test -[defsz] <path>]
        [-text [-ignoreCrc] <src> ...]
        [-touchz <path> ...]
        [-usage [cmd ...]]
```

HDFS DFS命令详解

- 查看目录结构
  - hdfs dfs -ls ------>查看当前目录结构
  - hdfs dfs -ls /input ------>查看指定目录（/input）结构
- 递归显示目录结构
  - hdfs dfs -ls -R ------>递归显示当前目录结构
  - hdfs dfs -ls -R /input ------>递归显示指定目录（/input）结构
- 创建目录
  - hdfs dfs -mkdir /input ------> 创建指定目录
  - hdfs dfs -mkdir -p /user/hdfs ------>递归创建指定目录

HDFS DFS命令详解

- 上传文件
  - hdfs dfs -put ./someWords.txt /input
- 移动文件
  - hdfs dfs -ls -R ------>递归显示当前目录结构
- 删除文件
  - hdfs dfs -rm /input/ someWords.txt
- 查看文件内容
  - hdfs dfs -cat /input/ someWords.txt
- 从本地文件系统复制文件到HDFS
  - hdfs dfs --copyFromLocal someWords_2.txt /input

- 修改文件权限（与Linux shell中类似）
  - hdfs dfs -chmod 777 /input/someWords.txt
- 修改属主
  - hdfs dfs -chown yarn /input/someWords.txt
- 修改属组
  - hdfs dfs -chgrp yarn /input/someWords.txt

HDFS的java访问接口

- org.apache.hadoop.fs.FileSystem
  - 是一个通用的文件系统API，提供了不同文件系统的统一访问方式。
- org.apache.hadoop.fs.Path
  - Hadoop文件系统中统一的文件或目录描述，类似于java.io.File对本地文件系统的文件或目录描述。
- org.apache.hadoop.conf.Configuration
  - 读取、解析配置文件(如core-site.xml/hdfs-default.xml/hdfs-site.xml等)，或添加配置的工具类
- org.apache.hadoop.fs.FSDataOutputStream
  - hadoop中数据输出流的统一封装
- org.apache.hadoop.fs.FSDataInputStream
  - hadoop中数据输入流的统一封装

Java访问HDFS的主要编程步骤

- 构建Configuration对象，读取并解析相关配置文件
  - Configuration conf=new Configuration();
- 获取特定文件系统实例fs（以HDFS文件系统实例）
  - FileSystem fs=FileSystem.get(new URI("hdfs://host1:9000"),conf,“hdfs");
- 通过文件系统实例fs进行文件操作(以删除文件实例)
  - fs.delete(new Path("/user/niuhy/someWords.txt"));

文件操作示例—前期准备

```
package com.briup.hdfs;

//此处略去import

public class HDFSTest {
        //文件系统实例
        FileSystem fileSystem;
        /**
         * 操作文件系统前，先获取FileSystem实例
         * @throws Exception
         */
        @Before
        public void before() throws Exception{
                Configuration conf=new Configuration();
                fileSystem = FileSystem.get(new URI("hdfs://192.168.150.128:9000"), conf, "hdfs");
        }
        /**
         * 操作完毕后，关闭FileSystem实例
         * @throws IOException
         */
        @After
        public void after() throws IOException{
                fileSystem.close();
        }
```

文件操作示例—创建删除目录

```
/**
 * 创建文件目录
 * @throws Exception
 */
@Test
public void mkDir() throws Exception{
        fileSystem.mkdirs(new Path("/user/hdfs"));
        System.out.println("目录创建成功！");
}
/**
 * 删除文件目录
 * @throws Exception
 */
@Test
public void deleteDir() throws Exception{
        fileSystem.delete(new Path("/user/hdfs"),true);
        System.out.println("目录删除成功！");
}
```

文件操作示例—查看文件信息

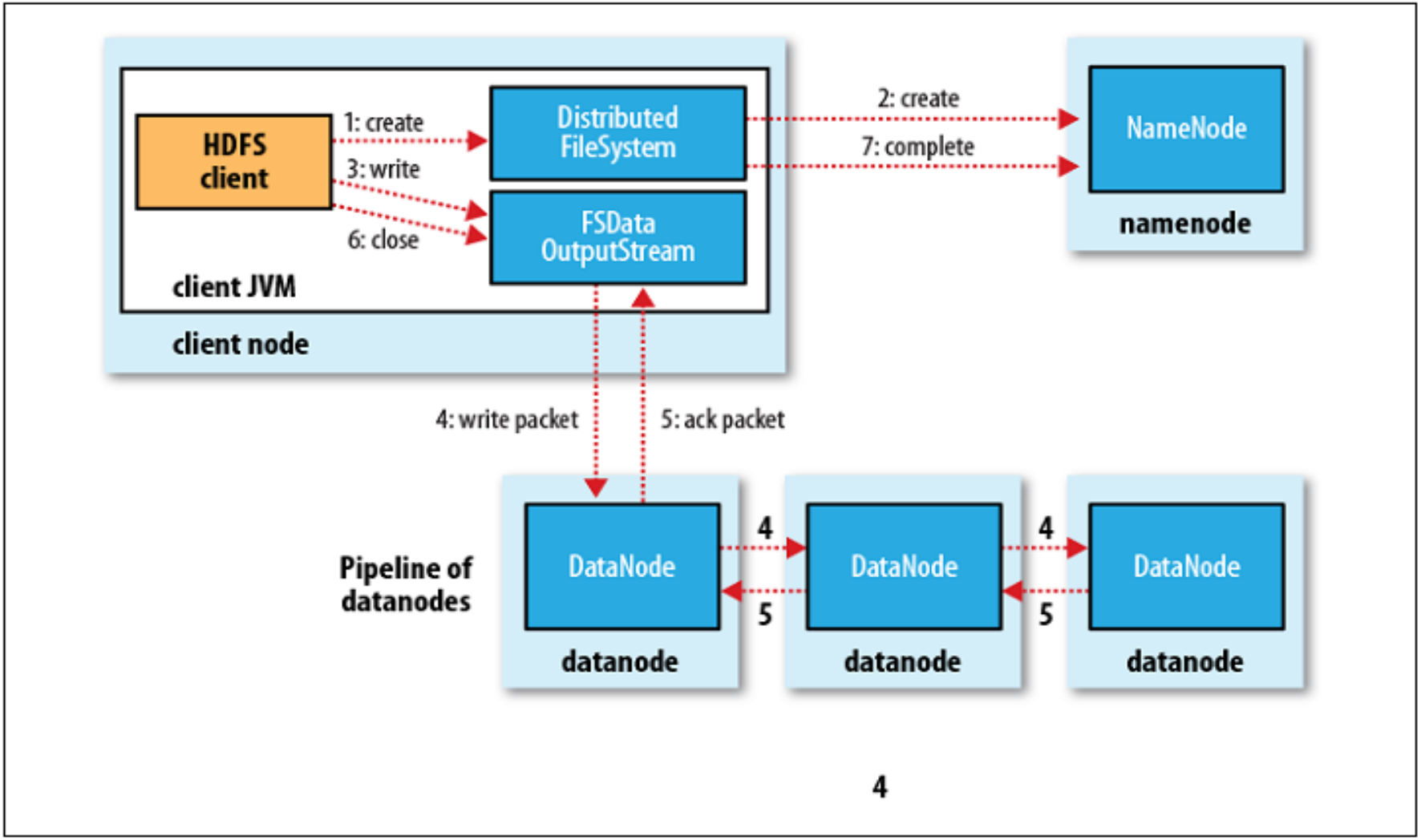

```
/**
 * 查看文件信息
 */
@Test
public void listDir() throws Exception {
    FileStatus[] filesStatus = fileSystem.listStatus(new Path("/"));
     System.out.println("============================");
     System.out.println("类型\t权限\t\t副本数\t拥有者\t路径");
     for(FileStatus fileStatus:filesStatus){
         printDir(fileStatus);
         if(fileStatus.isDirectory()){
             FileStatus[] filesStatus_2 = fileSystem.listStatus(fileStatus.getPath());
             for(FileStatus fileStatus_2:filesStatus_2){
                 printDir(fileStatus_2);
                 if(fileStatus.isDirectory()){
                     FileStatus[] filesStatus_3 = fileSystem.listStatus(fileStatus_2.getPath());
                     for(FileStatus fileStatus_3:filesStatus_3){
                         printDir(fileStatus_3);
                     }
                 }
             }
         }
     }
  }
}
```

```
/**
 * 打印文件信息工具类
 * @param fileStatus
 */
private void printDir(FileStatus fileStatus){
    String out="";
    out+=fileStatus.isDirectory()?"d\t":"-\t";
    out+=fileStatus.getPermission().toString()+"\t";
    out+=fileStatus.getReplication()+"\t";
    out+=fileStatus.getOwner()+"\t";
    out+=fileStatus.getPath().toString();
    System.out.println(out);
}
```

文件操作示例—读取和写入文件


```
        /**
         * 向HDFS写入数据
         */
        @Test
        public void writeSomeThing() throws Exception{
                FSDataOutputStream os = fileSystem.create(new Path("/user/hdfs/someWords.txt"), true);
                String words=" hello hadoop \n hello hdfs \n hello mapreduce \n hello yarn\n";
                os.writeUTF(words);
                os.close();
                System.out.println("文件写入完毕！");
        }
        /**
         * 从HDFS中读取数据        */
        @Test
        public void readSomeThing() throws Exception{
                FSDataInputStream in = fileSystem.open(new Path("/user/hdfs/someWords.txt"));
                System.out.println(in.readUTF());;
                in.close();
                System.out.println("文件读取完毕！");
        }
}
```

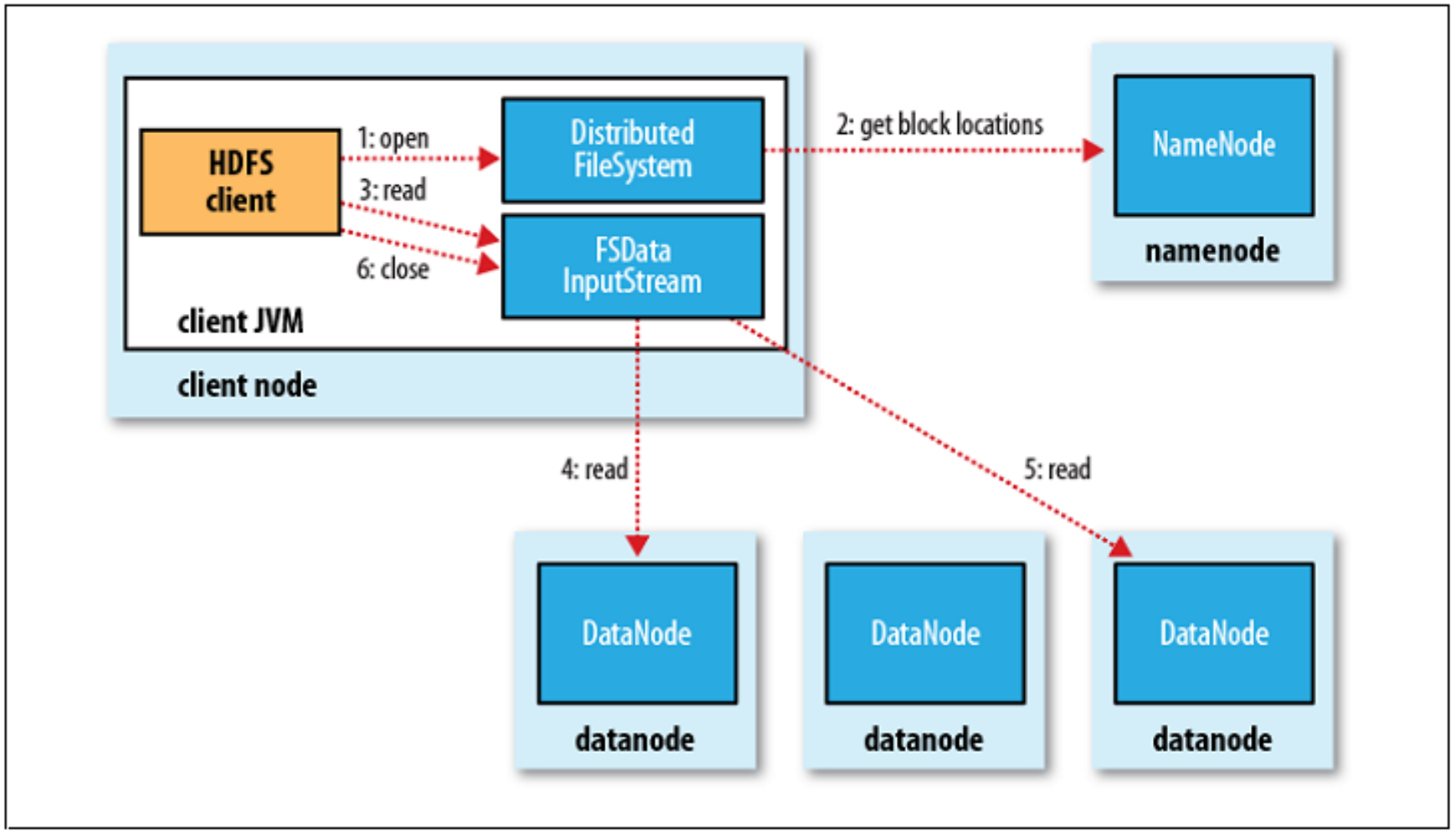
HDFS文件写入剖析图示



HDFS文件写入剖析

1. 客户端通过对象DistributedFileSystem的create方法创建文件
2. DistributedFileSystem创建一个RPC调用，申请在文件系统的命名空间中创建一个新的文件，此时文件中还没有相应的数据块。若权限验证通过，返回一个FSDataOutputStream对象。
3. 客户端开始使用FSDataOutputStream对象写入数据。其中， FSDataOutputStream 中包含一个DFSOutputStream对象，主要负责与datanode和namenode的通信。
4. DFSOutputStream将数据分成数据包，写入data queue。data queue由DataStreamer读取，并通知namenode分配DataNode，用来存储数据块(每块默认复制3块)。分配的DataNode放在一个pipeline里。Data Streamer将数据块写入pipeline中的第一个数据节点。第一个数据节点将数据块发送给第二个数据节点。第二个数据节点将数据发送给第三个数据节点。
5. DFSOutputStream为发出去的数据块保存了ack queue，等待pipeline中的数据节点告知数据已经写入成功。如果DataNode在写入的过程中失败，关闭pipeline，将ack queue中的数据块放入data queue的开始，当前的Block在已经写入的DataNode中被NameNode赋予新的标示，则错误节点重启后能够察觉其数据块是过时的，会被删除。失败的DataNode从pipeline中移除，另外的Block则写入pipeline中的另外两个DataNode。NameNode则被通知此Block是复制块数不足，将来会再创建第三份备份。
6. 当客户端结束写入数据，则调用stream的close函数。此操作将所有的数据块写入pipeline中的数据节点，并等待ack queue返回成功。
7. 最后通知NameNode写入完毕。

HDFS文件读取剖析图示



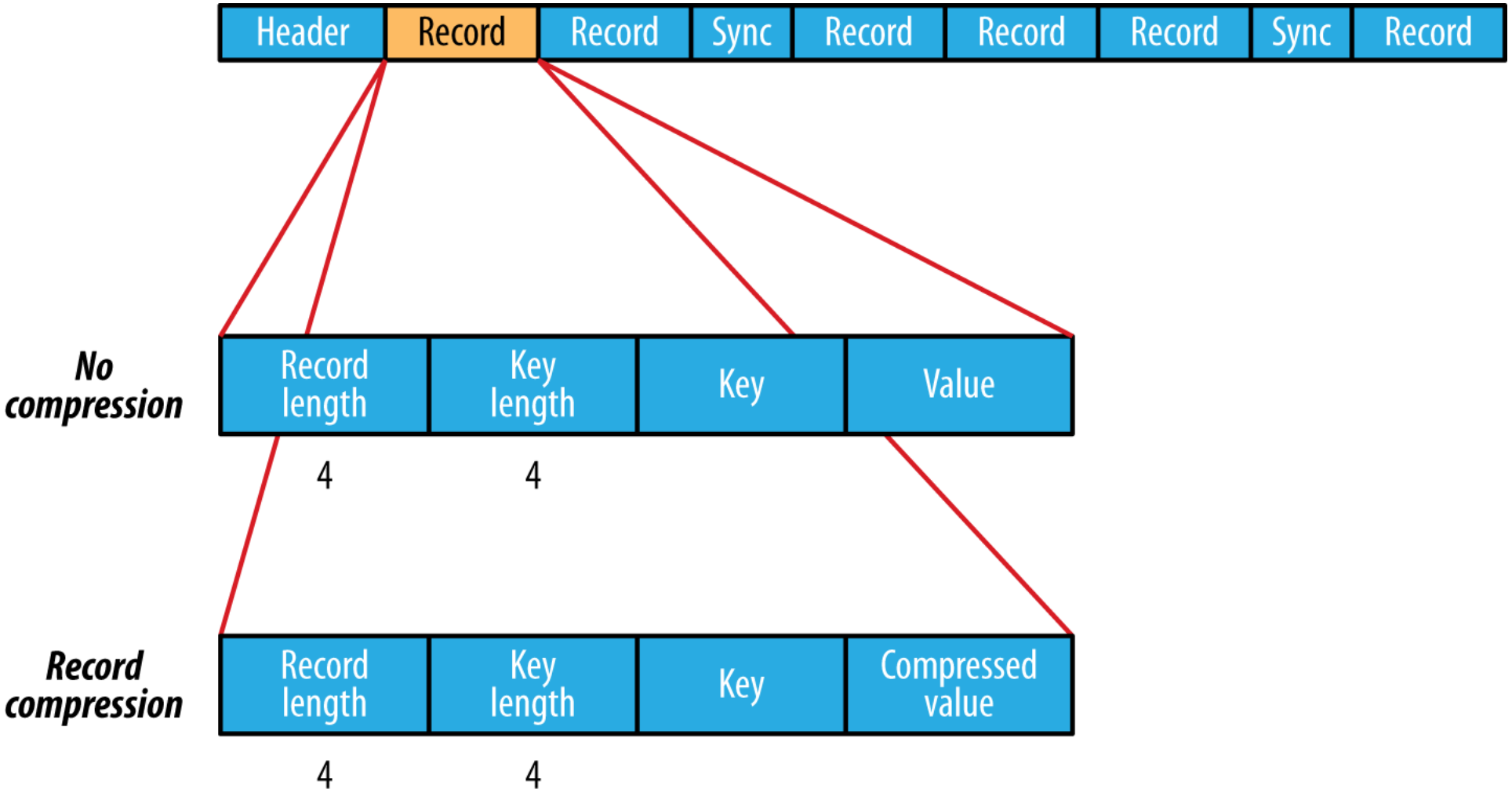
HDFS文件读取剖析过程

1. 客户端(client)用FileSystem(HDFS对应于DistributedFileSystem)实例的open()函数打开文件
2. FileSystem用RPC调用NameNode，得到文件的Blocks信息，对于每一个Block，NameNode返回保存Block的DataNode的地址，通过FSDataInputStream封装。 FSDataInputStream中进一步封装DFSInputStream对象，该对象主要用于与NameNode和DataNode通信。
3. 客户端调用FSDataInputStream的read()函数开始读取数据。
4. DFSInputStream连接保存此文件第一个Block的最近的DataNode，数据从DataNode读到客户端。在读取数据的过程中，如果客户端在与DataNode通信出现错误，则尝试连接包含此Block的下一个DataNode。失败的DataNode将被记录，以后不再连接。
5. 当此Block读取完毕时，DFSInputStream关闭和此DataNode的连接，然后连接此文件下一个Block的最近的DataNode。
6. 当客户端读取完毕数据的时候，调用FSDataInputStream的close函数。

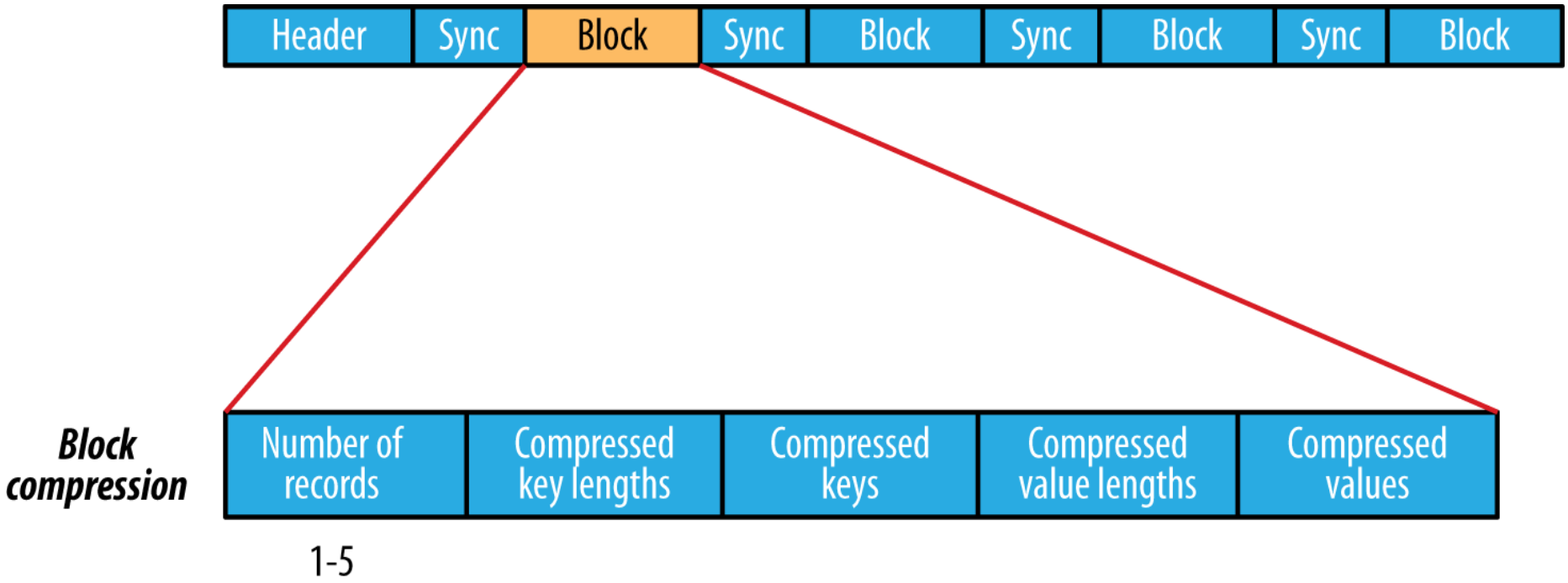
基于文件的数据结构--SequenceFile

- Hadoop用来存储二进制形式的key-value对而设计的一种平面文件(Flat File)
- 在SequenceFile文件中，每一个key-value被看做是一条记录(Record)
  - ✓ 其中，key和value要保证能被Serialization的
  - ✓ Hadoop使用自己的序列化格式：Writable类型
- 基于Record的压缩策略，可支持三种压缩类型(SequenceFile.CompressionType)
  - ✓ NONE: 对records不进行压缩
  - ✓ RECORD: 仅压缩每一个record中的value值
  - ✓ BLOCK: 将一个block中的所有records压缩在一起

SequenceFile存储格式



SequenceFile存储格式



SequenceFile的写操作

```
public class SequenceFileWriterDemo {
        private static  final String[] DATA={"One, two, buckle my shoe","Three, four, shut the door","Five, six, pick up sticks",
                        "Seven, eight, lay them straight","Nine, ten, a big fat hen"};

        public static void main(String[] args) throws IOException{
                Configuration conf=new Configuration();
                String uri="hdfs://192.168.150.128:9000/user/hdfs/numbers.seq";
                Path path=new Path(uri);
                IntWritable key=new IntWritable();
                Text value=new Text();
                //获取SequenceFile的写入对象
                SequenceFile.Writer writer=SequenceFile.createWriter(conf, SequenceFile.Writer.file(path),
                                SequenceFile.Writer.keyClass(key.getClass()),
                                SequenceFile.Writer.valueClass(value.getClass()));
                for(int i=0;i<100;i++){
                        key.set(100-i);
                        value.set(DATA[i%DATA.length]);
                        System.out.printf("[%s]\t%s\t%s\n",writer.getLength(),key,value);
                        //写入数据
                        writer.append(key, value);
                }
                writer.close();
        }
}
```

SequenceFile的读操作

```
public class SequenceFileReaderDemo {
        public static void main(String[] args) throws IOException{
                Configuration conf=new Configuration();
                String uri="hdfs://192.168.150.128:9000/user/hdfs/numbers.seq";
                Path path=new Path(uri);
                //获取SequenceFile的读取对象
                SequenceFile.Reader reader=new SequenceFile.Reader(conf, SequenceFile.Reader.file(path));

                IntWritable key=new IntWritable();
                Text value=new Text();
                long position=reader.getPosition();
                //迭代的读取数据
                while(reader.next(key, value)){
                        String syncSeen = reader.syncSeen()?"*":"";
                        System.out.printf("[%s%s]\t%s\t%s\n",position,syncSeen,key,value);
                }

                reader.close();
        }
}
```

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop之MapReduce

Brighten Your Way And Raise You Up.

时代背景—爆炸性增长的Web规模数据量

- Google从2004年每天处理100TB数据到2008年每天处理20PB
- 2009年eBays数据仓库，一个有2PB用户数据，另一个6.5PB用户数据包含170TB记录且每天增长150GB个记录；Facebook：2.5PB用户数据，每天增加15TB
- 世界最大电子对撞机每年产生15PB(1千5百万GB)数据
- 2015年落成的世界最大观天望远镜主镜头像素为3.2G，每年将产生6PB天文图像数据；
- 欧洲生物信息研究中心(EBI)基因序列数据库容量已达5PB；中国深圳华大基因研究所成为全世界最大测序中心，每天产生300GB基因序列数据（每年100TB）

时代背景—超大的计算量/计算复杂度

- 用SGI工作站进行电影渲染时，每帧一般需要1～2小时
  - 一部2小时的电影渲染需要：

    2小时x3600秒x24帧x(1~2小时)/24小时=20~40年!
- 特殊场景每帧可能需要60个小时(影片“星舰骑兵”中数千只蜘蛛爬行的场面),用横向4096象素分辨率进行渲染时，如果以每帧60个小时的速度，则1秒的放映量（24帧）需要60天的渲染时间,1分钟则需要100年！
- 世界著名的数字工作室Digital Domain公司用了一年半的时间，使用了300多台SGI超级工作站，50多个特技师一天24小时轮流制作「泰坦尼克号」中的电脑特技

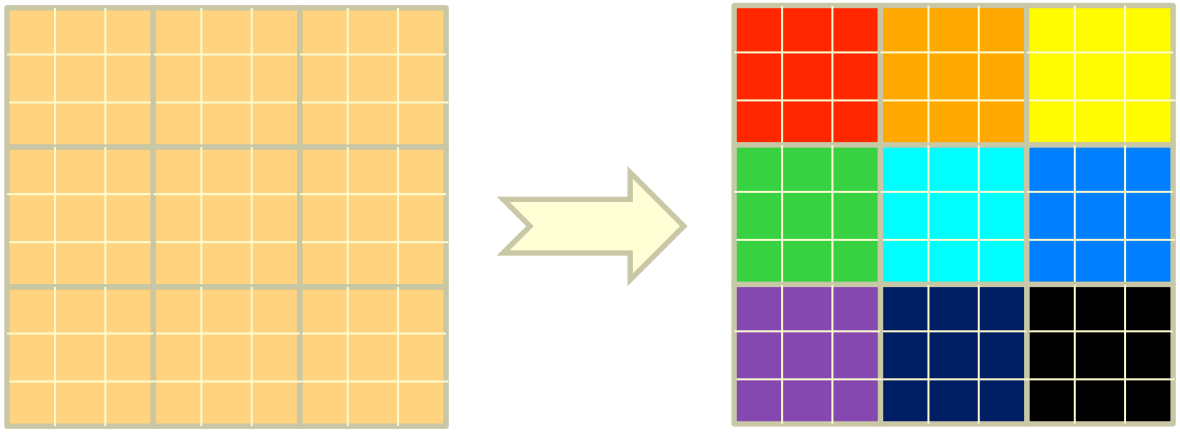
并行计算大势所趋

- 海量数据及其处理已经成为现实世界的急迫需求
- 数据的处理能力大幅落后于数据的增长
  - 100TB数据顺序读一遍需要多少时间？
    - ✓ 设硬盘读取访问速率128MB/秒
    - ✓ 1TB/128MB 约2.17小时
    - ✓ 100TB/128MB = 217小时 = 9天！
    - ✓ 即使用百万元高速磁盘阵列(800MB/s),仍需1.5天！
- 需要寻求更为有效的数据密集型并行计算方法！！

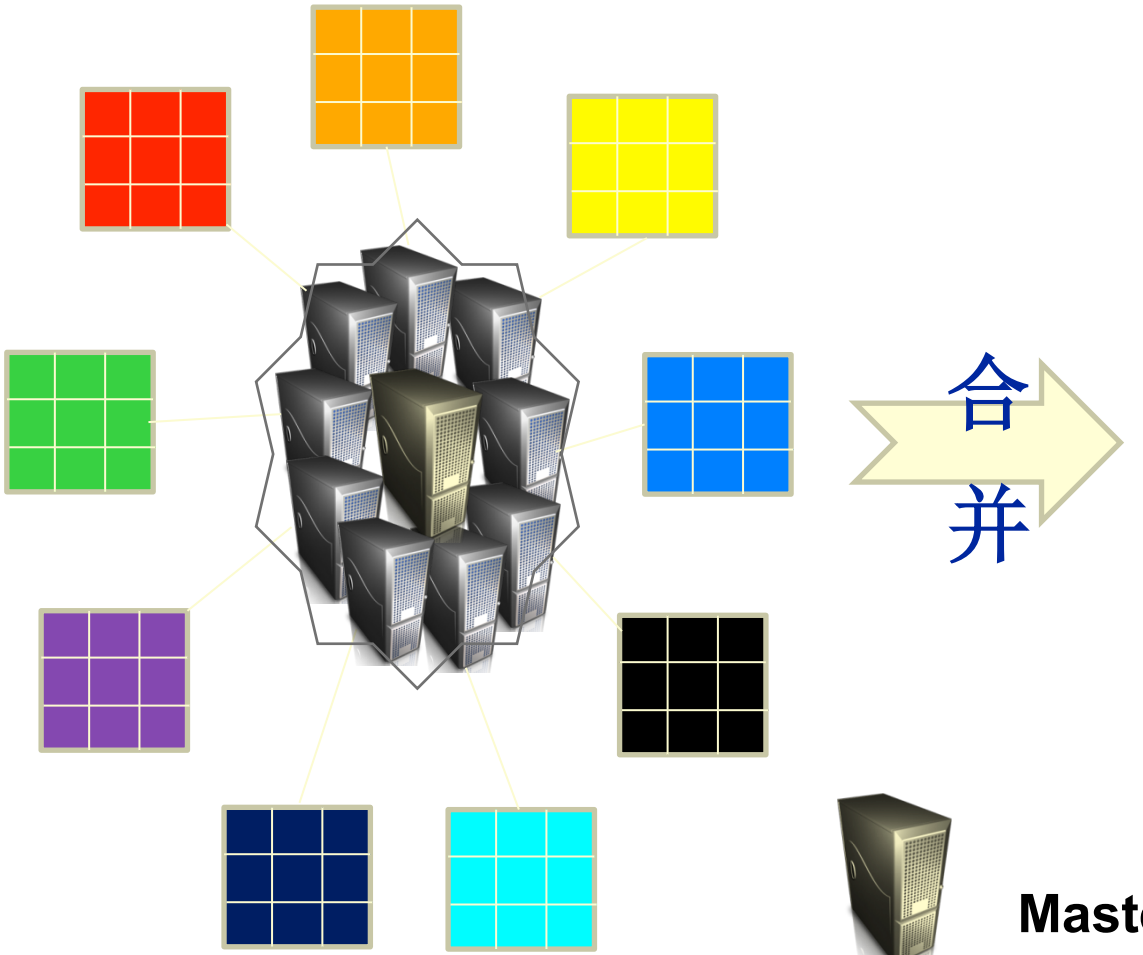
大数据的并行计算

- 一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是并行计算

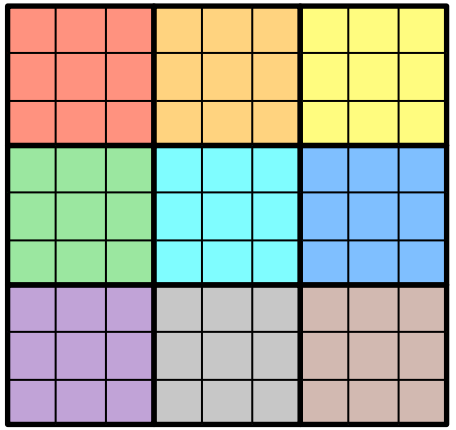
例如：假设有一个巨大的2维数据需要处理(比如求每个元素的开立方)，其中对每个元素的处理是相同的，并且数据元素间不存在数据依赖关系，可以考虑不同的划分方法将其划分为子数组，由一组处理器并行处理



大数据的并行计算



合并



Master：负责划分和分配任务

Worker：负责数据块计算

为什么需要MapReduce？

- 并行计算技术和并行程序设计的复杂性
  - 依赖于不同类型的计算问题、数据特征、计算要求、和系统构架，并行计算技术较为复杂,程序设计需要考虑数据划分，计算任务和算法划分，数据访问和通信同步控制,软件开发难度大,难以找到统一和易于使用的计算框架和编程模型与工具
- MapReduce是目前业界和学界公认的最为有效和最易于使用的面向海量数据并行处理技术
- Google,Yahoo!,IBM,Amazon,百度、淘宝、腾讯等国内外公司普遍使用MapReduce
  - Google:超过7千个程序基于MapReduce开发

- 2004年，Google在OSDI国际会议上发表了一篇论文“MapReduce：Simplified Data Processing on Large Clusters”，公布了MapReduce的基本原理和主要设计思想。
- Google公司用MapReduce重新改写了其整个搜索引擎中的Web文档索引处理

VS

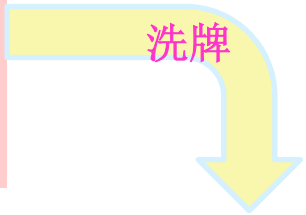
- 在Google发表了文章后，Doug Cutting，2004年，开源项目Lucene(搜索索引程序库)和Nutch(搜索引擎)的创始人，发现MapReduce正是其所需要的解决大规模分布数据处理的重要技术，因而模仿Google MapReduce，基于Java设计出了称为Hadoop的开源MapReduce，该项目成为Apache下最重要项目
- Yahoo!是Hadoop联盟中最大的支持者，目前大量使用了Hadoop集群

- 面向大规模数据并行处理
- 基于集群的高性能并行计算平台
- 并行程序设计模型和方法
- 并行程序开发和运行框架

MapReduce编程模型

- MapReduce借鉴了函数式程序设计语言Lisp中的思想，定义了如下的Map和Reduce两个抽象的编程接口，由用户去编程实现

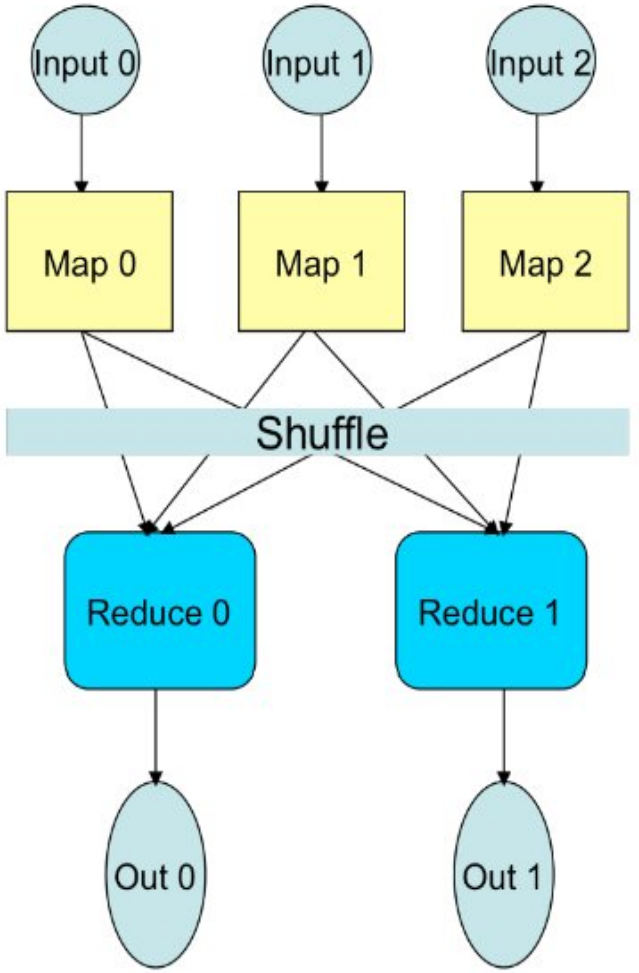
- Map函数：(k1; v1)-->[(k2; v2)]
  - 输入：键值对(k1;v1)表示的数据
  - 处理：文档数据记录(如文本文件中的一行)以键值对的形式传入map函数，处理完之后以另一种键值对的形式输出中间处理结果[k2;v2]
  - 输出：键值对[k2;v2]表示的一组中间数据



- Reduce函数：(k2; [v2])-->[(k3; v3)]
  - 输入：map输出的一组键值对[(k2; v2)]将被进行合并处理将同样主键下的不同数值合并到一个列表[v2]中，故reduce的输入为(k2; [v2])
  - 处理：对传入的中间结果列表数据进行某种整理或进一步的处理,并输出最终的某种键值对形式的结果[(k3;v3)]
  - 输出：键值对[k3;v3]表示的最终数据

MapReduce编程模型

- 各个map函数对所划分的数据并行处理，从不同的输入数据产生不同的中间结果输出
- 各个reduce也各自并行计算，各自负责处理不同的中间结果数据集合
- 进行reduce处理之前,必须等到所有的map函数做完，因此,在进入reduce前需要有一个同步障(barrier);这个阶段也负责对map的中间结果数据进行收集整理(aggregation & shuffle)处理,以便reduce更有效地计算最终结果
- 最终汇总所有reduce的输出结果即可获得最终结果



文档词频统计WordCount实例—分析

- 设有4组原始文本数据：
  - Text 1: the weather is good　　Text 2: today is good
  - Text 3: good weather is good　　Text 4: today has good weather

文档词频统计WordCount实例—分析

- 使用四个map节点：
  - map节点1:
    - 输入：(text1, “the weather is good”)
    - 输出：(the, 1), (weather, 1), (is, 1), (good, 1)
  - map节点2:
    - 输入：(text2, “today is good”)
    - 输出：(today, 1), (is, 1), (good, 1)
  - map节点3:
    - 输入：(text3, “good weather is good”)
    - 输出：(good, 1), (weather, 1), (is, 1), (good, 1)
  - map节点4:
    - 输入：(text3, “today has good weather”)
    - 输出：(today, 1), (has, 1), (good, 1), (weather, 1)

文档词频统计WordCount实例—分析

- 使用一个reduce节点：
  - reduce节点:
    - ✓ 输入：(good, 1), (good, 1), (good, 1), (good, 1), (good, 1), (has, 1), (is,1), (is,1), (is, 1), (the, 1), (today, 1), (today, 1), (weather, 1), (weather,1), (weather, 1)
    - ✓ 输出：(good, 5), (has, 1), (is, 3), (the, 1), (today, 2), (weather, 3)

结果：

good: 5

is: 3

has:1

the:1

today:2

weather: 3

文档词频统计WordCount实例—代码实现

```
/**
 *通过继承org.apache.hadoop.mapreduce.Mapper编写自己的Mapper
 */
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
        private final static IntWritable one=new IntWritable(1); //统计使用变量
        private Text word=new Text(); //单词变量
        /**
         * key:当前读取行的偏移量
         * value：当前读取的行
         * context:map方法执行时上下文
         */
        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context)
                throws IOException, InterruptedException {
            // TODO Auto-generated method stub
            StringTokenizer words=new StringTokenizer(value.toString());
            while(words.hasMoreTokens()){
                word.set(words.nextToken());
                context.write(word, one);
            }
        }
}
```

文档词频统计WordCount实例—代码实现

```
/**
 * 通过继承org.apache.hadoop.mapreduce.Reducer编写自己的Reducer
 */
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>{

        /**
         * key:待统计的word
         * values:待统计word的所有统计标识
         * context:reduce方法执行时的上下文
         */
        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,Reducer<Text, IntWritable, Text, IntWritable>.Context
 context) throws IOException, InterruptedException {
                // TODO Auto-generated method stub
                int count=0;
                for(IntWritable one:values){
                        count+=one.get();
                }
                context.write(key, new IntWritable(count));
        }
}
```

文档词频统计WordCount实例—代码实现

```
/**
 * WordCount作业调度的驱动程序 *
 */
public class WordCountDriver {
        public static void main(String[] args) throws Exception {
//                构建新的作业
                Configuration conf=new Configuration();
                Job job = Job.getInstance(conf, "Word Count");
                job.setJarByClass(WordCountDriver.class);
//                设置Mapper和Reducer函数
                job.setMapperClass(WordCountMapper.class);
                job.setReducerClass(WordCountReducer.class);
//                设置输出格式
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
//                设置输入和输出目录
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
//                提交作业执行
                System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

文档词频统计实例—程序运行

- 前期准备
  - 将程序打成jar包：wordcount.jar
  - 准备好Text 1-4文件
- 运行
  - yarn jar wordcount.jar com.briup.WordCount input output

辅助类Tool和ToolRunner

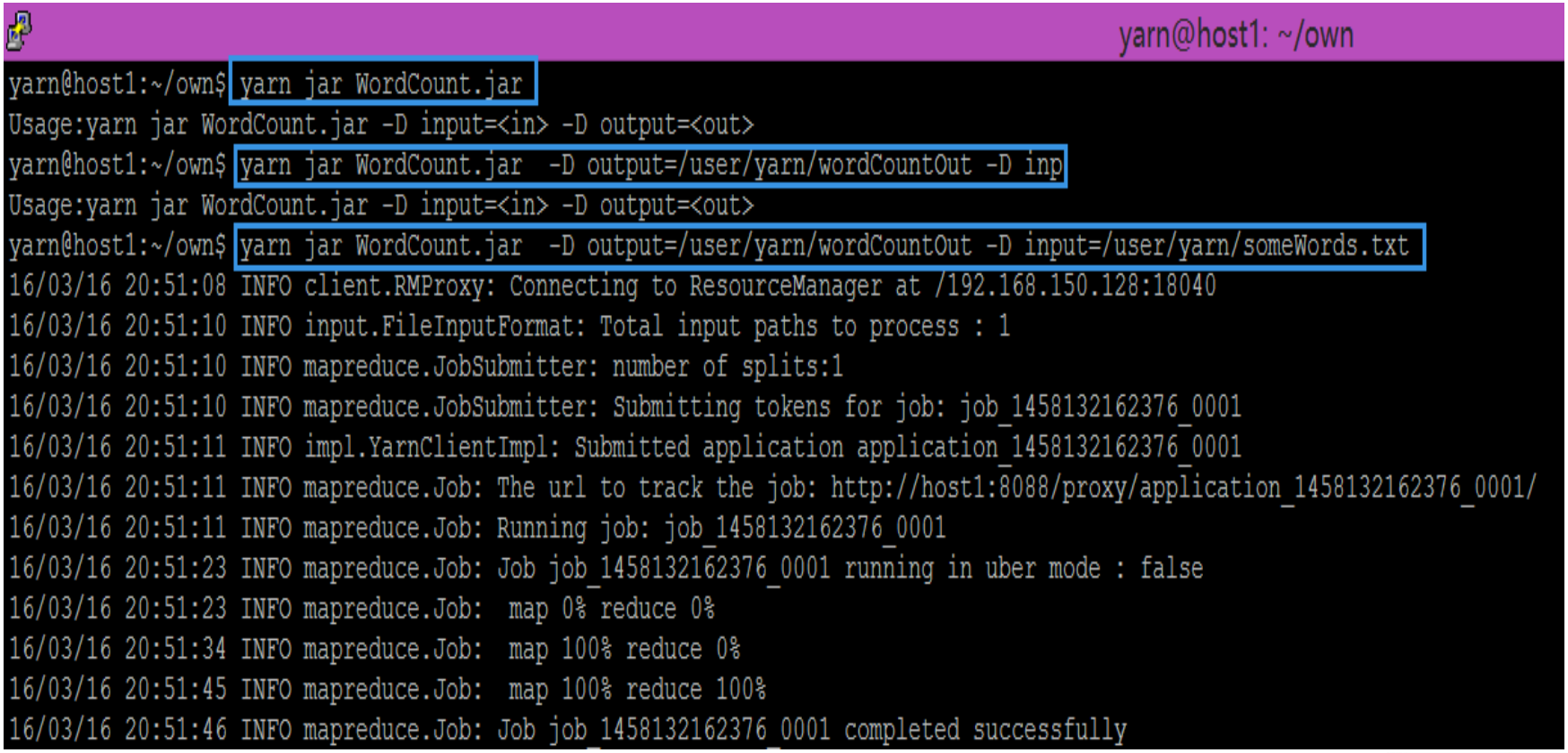
```
public class WordCountToolDriver extends Configured implements Tool{
        @Override
        public int run(String[] args) throws Exception {
//                      构建新的作业
                        Configuration conf=getConf();
                        Job job = Job.getInstance(conf, "Word Count");
                        job.setJarByClass(WordCountDriver.class);
                        job.setMapperClass(WordCountMapper.class);                      //设置Mapper函数
                        job.setReducerClass(WordCountReducer.class);                    //设置Reducer函数
                        job.setOutputKeyClass(Text.class);                      //设置输出格式
                        job.setOutputValueClass(IntWritable.class);             //设置输出格式
//                      设置输入和输出目录
                        String inputPath=conf.get("input");
                        String outputPath=conf.get("output");
                        if(inputPath==null||outputPath==null){
                                System.out.println("Usage:yarn jar WordCount.jar -D input=<in> -D output=<out>");
                                return 1;
                        }
                        FileInputFormat.addInputPath(job, new Path(inputPath));
                        FileOutputFormat.setOutputPath(job, new Path(outputPath));              //提交作业执行
                        return job.waitForCompletion(true)?0:1;                 //提交作业执行
        }
        public static void main(String[] args) throws Exception{
                        int status = ToolRunner.run(new WordCountToolDriver(), args);
                        System.exit(status);
        }
}
```

Table 6-1. GenericOptionsParser and ToolRunner options

Option	Description
`-D property=value`	Sets the given Hadoop configuration property to the given value. Overrides any default or site properties in the configuration and any properties set via the `-conf` option.
`-conf filename ...`	Adds the given files to the list of resources in the configuration. This is a convenient way to set site properties or to set a number of properties at once.
`-fs uri`	Sets the default filesystem to the given URI. Shortcut for `-D fs.defaultFS=uri`.
`-jt host:port`	Sets the YARN resource manager to the given host and port. (In Hadoop 1, it sets the jobtracker address, hence the option name.) Shortcut for `-D yarn.resourcemanager.address=host:port`.
`-files file1,file2,...`	Copies the specified files from the local filesystem (or any filesystem if a scheme is specified) to the shared filesystem used by MapReduce (usually HDFS) and makes them available to MapReduce programs in the task's working directory. (See "Distributed Cache" on page 274 for more on the distributed cache mechanism for copying files to machines in the cluster.)
`-archives archive1,archive2,...`	Copies the specified archives from the local filesystem (or any filesystem if a scheme is specified) to the shared filesystem used by MapReduce (usually HDFS), unarchives them, and makes them available to MapReduce programs in the task's working directory.
`-libjars jar1,jar2,...`	Copies the specified JAR files from the local filesystem (or any filesystem if a scheme is specified) to the shared filesystem used by MapReduce (usually HDFS) and adds them to the MapReduce task's classpath. This option is a useful way of shipping JAR files that a job is dependent on.

文档词频统计实例—程序运行

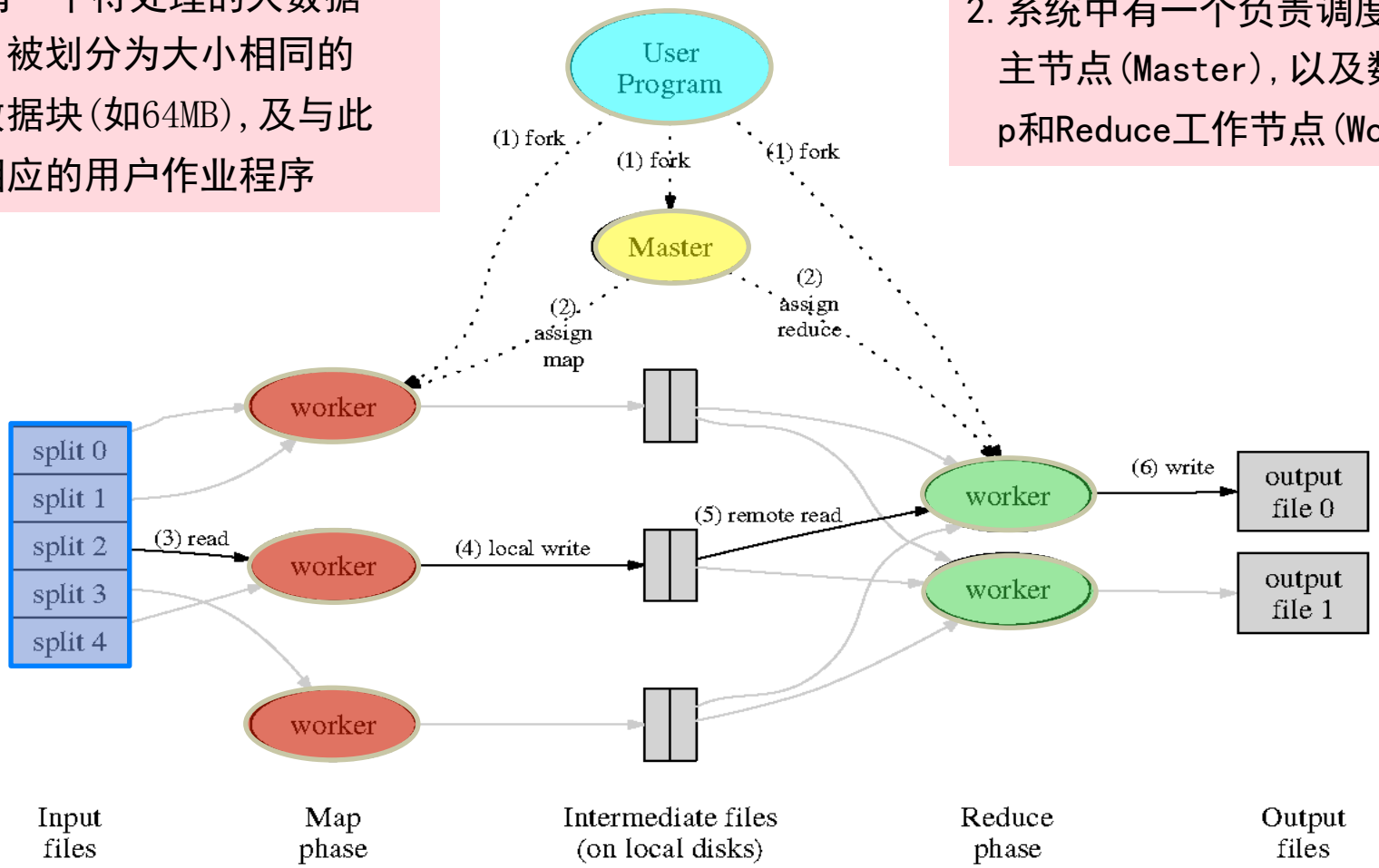
- 运行
  - yarn jar wordcount.jar com.briup.WordCount -D input=<input> -Doutput=<output>
- 运行效果如下：



MapReduce并行处理的基本过程

1. 有一个待处理的大数据，被划分为大小相同的数据块(如64MB)，及与此相应的用户作业程序

2. 系统中有一个负责调度的主节点(Master)，以及数据Map和Reduce工作节点(Worker)

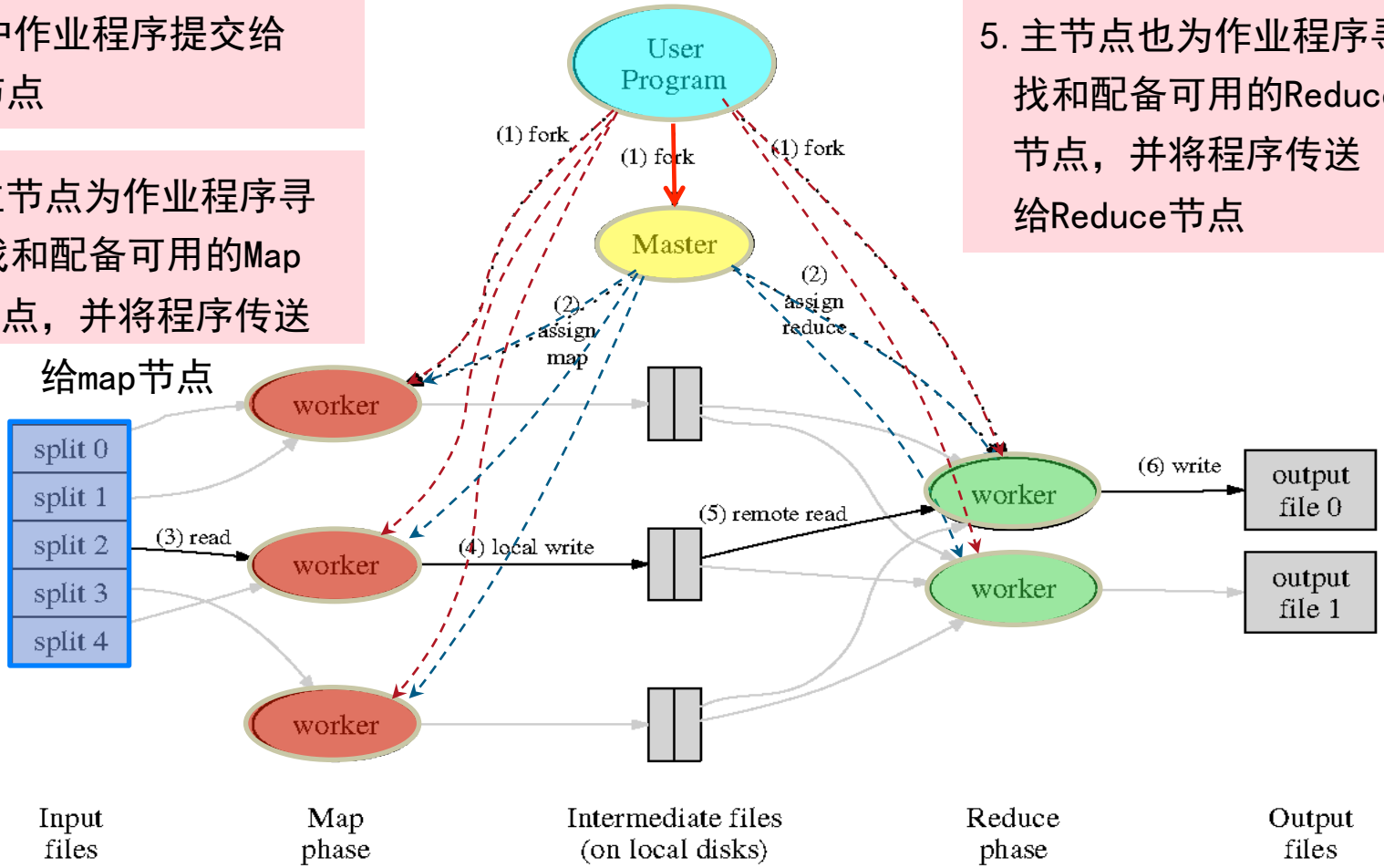


MapReduce并行处理的基本过程

3. 用户作业程序提交给主节点

4. 主节点为作业程序寻找和配备可用的Map节点，并将程序传送给map节点

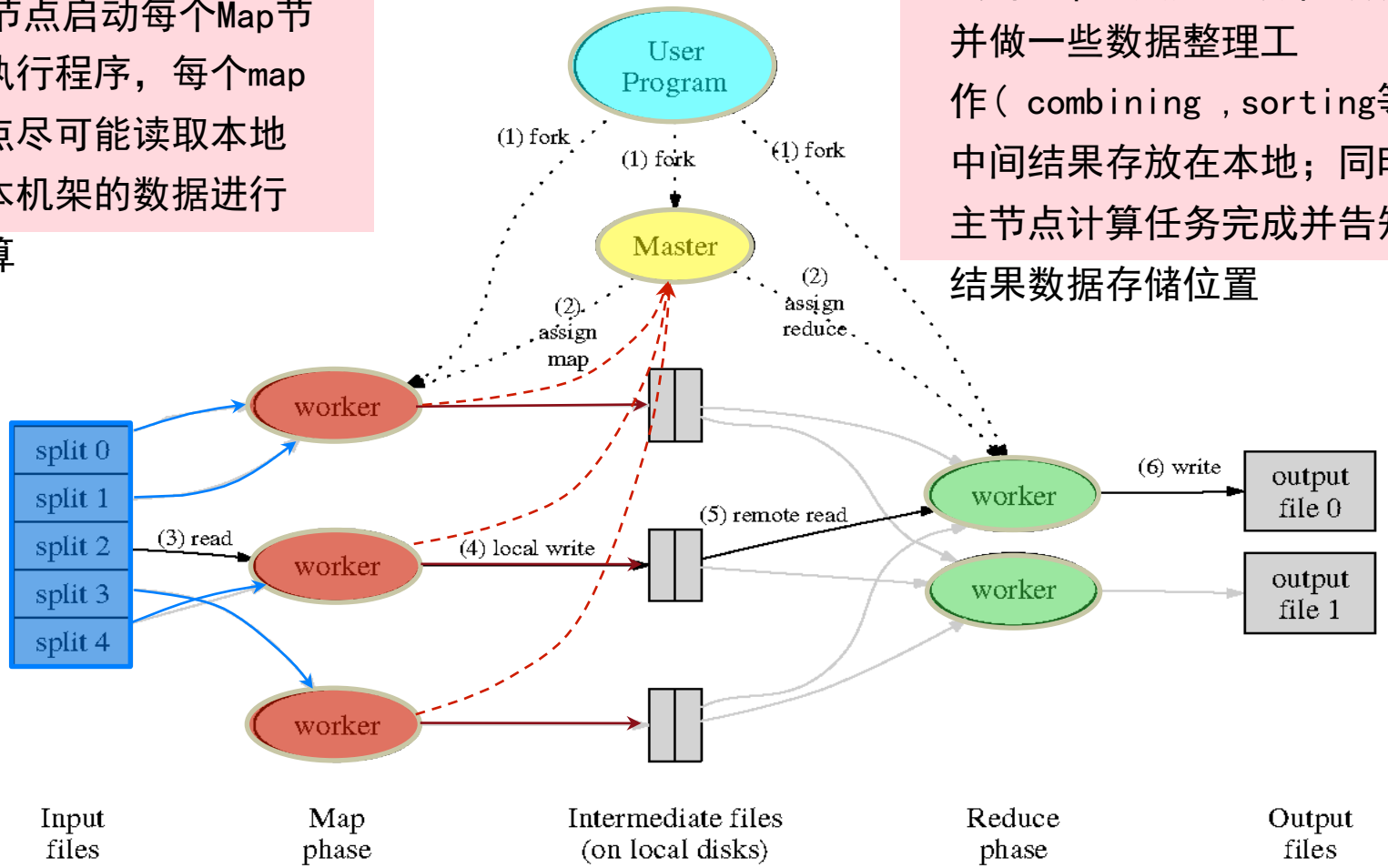
5. 主节点也为作业程序寻找和配备可用的Reduce节点，并将程序传送给Reduce节点



MapReduce并行处理的基本过程

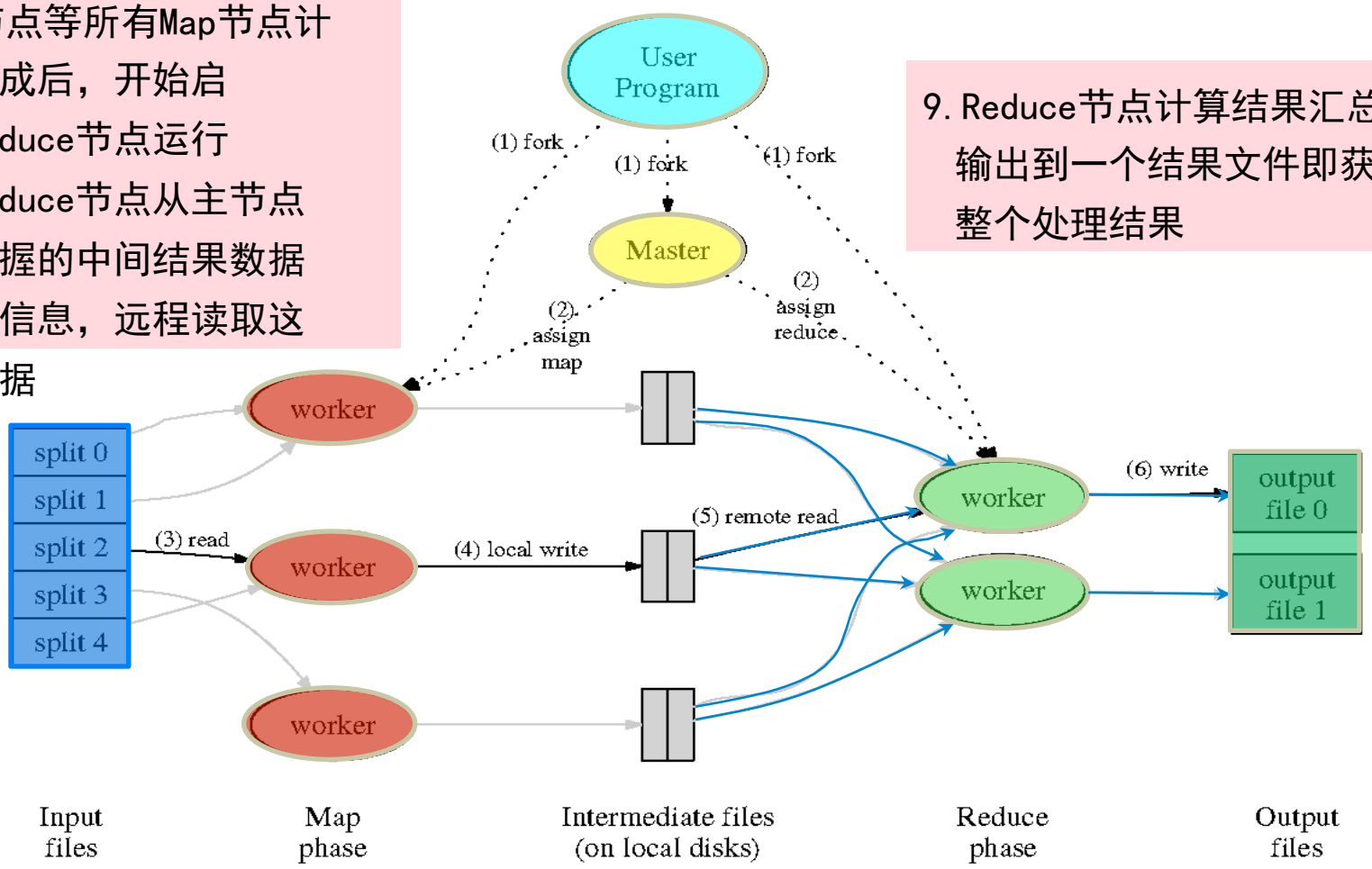
6. 主节点启动每个Map节点执行程序，每个map节点尽可能读取本地或本机架的数据进行计算

7. 每个Map节点处理读取的数据块，并做一些数据整理工作（combining，sorting等）并将中间结果存放在本地；同时通知主节点计算任务完成并告知中间结果数据存储位置



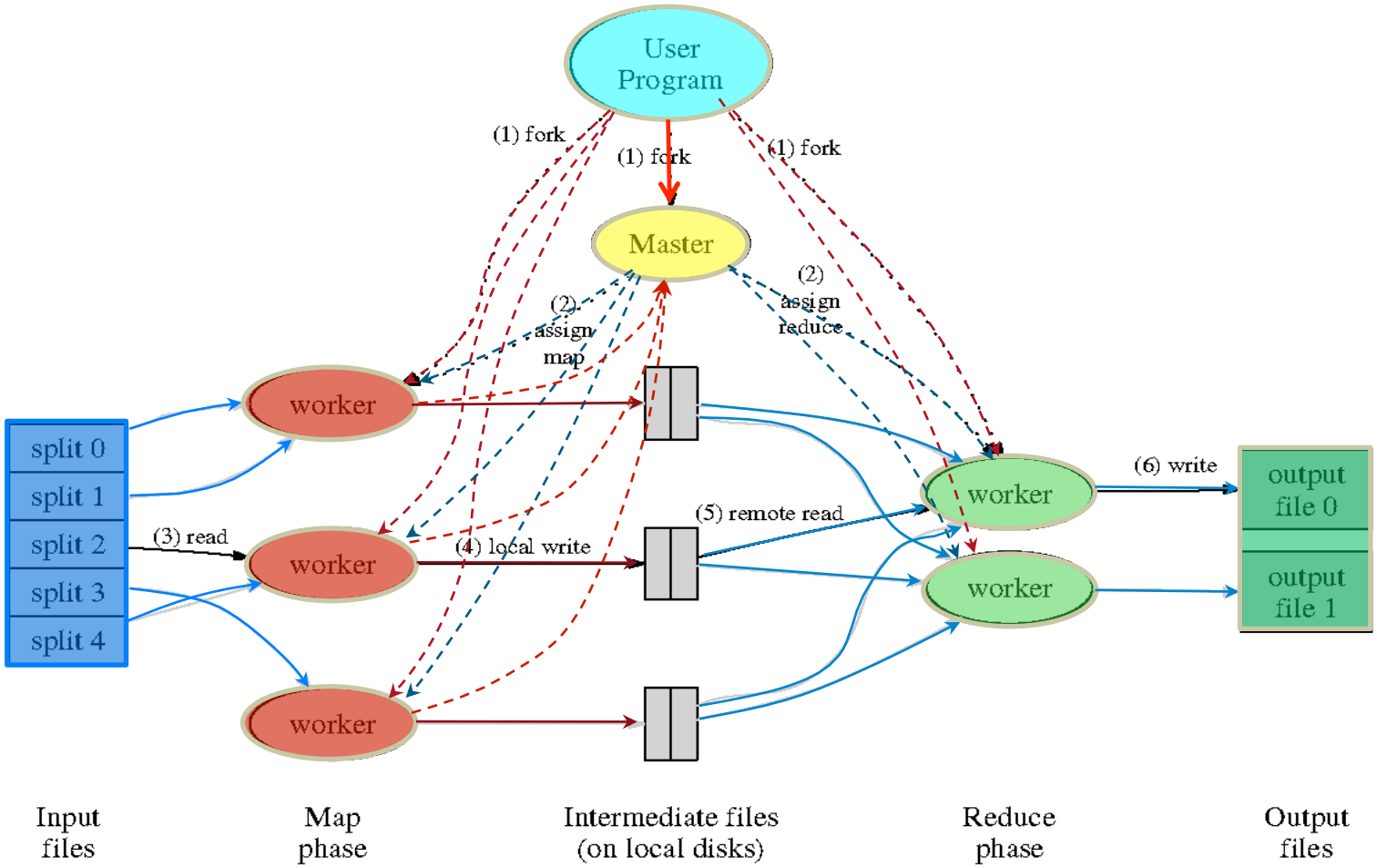
MapReduce并行处理的基本过程

8. 主节点等所有Map节点计算完成后，开始启动Reduce节点运行；Reduce节点从主节点所掌握的中间结果数据位置信息，远程读取这些数据



9. Reduce节点计算结果汇总输出到一个结果文件即获得整个处理结果

MapReduce并行处理的基本过程—回顾



杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop之MapReduce进阶

Brighten Your Way And Raise You Up.

序列化

- 序列化的概念
  - 序列化：将结构化对象转化为字节流以便在网络上传输或写入磁盘持久存储的过程
  - 反序列化：将字节流转回结构化对象的逆过程
- 应用领域
  - 进程间通信
  - 永久存储
- 序列化格式的特点
  - 紧凑------>充分利用网络带宽资源和存储空间
  - 快速------>减少序列化和反序列化的性能开销
  - 可扩展------>兼容老格式的消息
  - 互操作------>支持不同的语言
- 序列化实现
  - Java自己的序列化机制----->java.io.Serializable
  - Hadoop自己的序列化机制----->org.apache.hadoop.io.Writable

序列化的常用接口

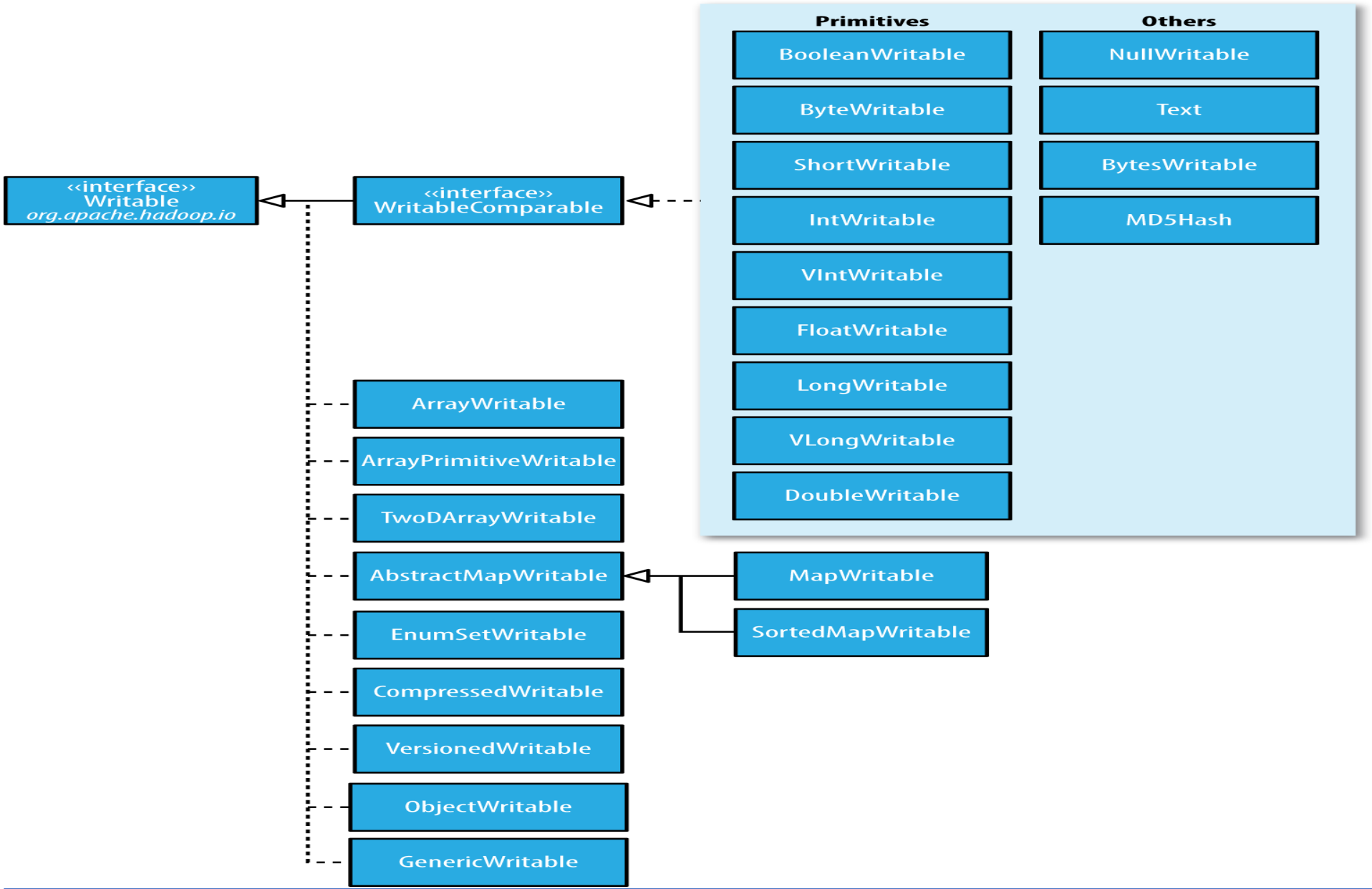
- org.apache.hadoop.io.Writable
  - 序列化的对象需实现的一个接口
  - 该接口包含两个抽象方法：

Method Summary	
void	readFields(DataInput in) Deserialize the fields of this object from in.
void	write(DataOutput out) Serialize the fields of this object to out.

  - Hadoop的MapReduce框架中key或value均需实现这个接口
- org.apache.hadoop.io.WritableComparable
  - 继承自上述Writable接口和java.lang.Comparable接口
  - Hadoop的MapReduce框架中的key必须实现这个接口
- org.apache.hadoop.io.RawComparator
  - 继承子java.util.Comparator
  - 一个抽象方法：可在数据流中比较对象

Method Summary	
int	compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) Compare two objects in binary.

Hadoop自带Writable类的层次结构图



Java基本数据类型的Writable类

- Writable类对Java基本数据类型提供了封装
  - short和char除外，可以使用IntWritable存储
  - 所有的封装包含get()和set()两方法用于读取或设置封装的值

Java 基本类型	Writable 实现	序列化大小(字节)
boolean	BooleanWritable	1
byte	ByteWritable	1
Short	ShortWritable	2
int	IntWritable	4
	VintWritable	1~5
float	FloatWritable	4
long	LongWritable	8
	VlongWritable	1～9
double	DoubleWritable	8

实现定制的Writable类型——part_1


```
public class TextPairWritable implements WritableComparable<TextPairWritable>{
    private IntWritable id;
    private Text name;
    private Text words;

    public TextPairWritable(){
        set(new IntWritable(), new Text(), new Text());
    }
    public TextPairWritable(int id,String name,String words){
        set(new IntWritable(id), new Text(name), new Text(words));
    }
    public TextPairWritable(IntWritable id,Text name,Text words){
        set(id, name, words);
    }

    public void set(IntWritable id,Text name,Text words){
        this.id=id;
        this.name=name;
        this.words=words;
    }

    public void set(int id,String name,String words){
        this.id.set(id);;
        this.name.set(name);
        this.words.set(words);
    }

    public IntWritable getId() {
        return id;
    }
    public Text getName() {
        return name;
    }
    public Text getWords() {
        return words;
    }
    @Override
    public String toString() {
        return "TextPairWritable [id=" + id + ", name=" + name + ", words=" + words + "]";
    }
```

实现定制的Writable类型——part_2

```
    /**
     * Writable接口中反序列化方法
     */
    @Override
    public void readFields(DataInput input) throws IOException {
        id.readFields(input);
        name.readFields(input);
        words.readFields(input);
    }

    /**
     * Writable接口中序列化方法
     */
    @Override
    public void write(DataOutput out) throws IOException {
        id.write(out);
        name.write(out);
        words.write(out);
    }

    /**
     * Comparable接口中的比较方法
     */
    @Override
    public int compareTo(TextPairWritable o) {
        int result=this.id.compareTo(o.id);
        if(result!=0){
            return result;
        }else{
            result=this.name.compareTo(o.name);
            if(result!=0){
                return result;
            }else{
                result=this.words.compareTo(o.words);
                return result;
            }
        }
    }
}
```

实现定制的Writable类型——测试类

```
public class Test {
    public static void main(String[] args) throws Exception {
        Configuration conf=new Configuration();
        FileSystem fs = FileSystem.get(URI.create("hdfs://192.168.150.128:9000"), conf, "hdfs");
        FSDataOutputStream out = fs.create(new Path("/user/hdfs/testWritable"), true);

        for(int i=0;i<50;i++){
            TextPairWritable t=new TextPairWritable(i, "name_"+i, "hello_"+i);
            t.write(out);
        }
        out.close();
        System.out.println("书写完毕!");
        FSDataInputStream input = fs.open(new Path("/user/hdfs/testWritable"));
        TextPairWritable t=new TextPairWritable();
        for(int i=0;i<50;i++){
            t.readFields(input);
            System.out.println(t.getId()+"\t"+t.getName()+"\t"+t.getWords());
        }
        input.close();
        System.out.println("读取完毕!");
    }
}
```

MapReduce的类型配置

Table 8-1. Configuration of MapReduce types in the new API

Property	Job setter method	Input types		Intermediate types		Output types	
		K1	V1	K2	V2	K3	V3
Properties for configuring types: 用于配置类型的属性							
mapreduce.job.inputformat.class	setInputFormatClass()	•	•				
mapreduce.map.output.key.class	setMapOutputKeyClass()			•			
mapreduce.map.output.value.class	setMapOutputValueClass()				•		
mapreduce.job.output.key.class	setOutputKeyClass()					•	
mapreduce.job.output.value.class	setOutputValueClass()						•
Properties that must be consistent with the types: 必须与类型一致的属性							
mapreduce.job.map.class	setMapperClass()	•	•	•	•		
mapreduce.job.combine.class	setCombinerClass()			•	•		
mapreduce.job.partitioner.class	setPartitionerClass()			•	•		
mapreduce.job.output.key.comparator.class	setSortComparatorClass()			•			
mapreduce.job.output.group.comparator.class	setGroupingComparatorClass()			•			
mapreduce.job.reduce.class	setReducerClass()			•	•	•	•
mapreduce.job.outputformat.class	setOutputFormatClass()					•	•

默认的MapReduce程序

```
/**
 * 没有指定Mapper和Reducer的最小作业配置
 */
public class MinimalMapReduce {
        public static void main(String[] args) throws Exception{
//                      构建新的作业
                        Configuration conf=new Configuration();
                        Job job = Job.getInstance(conf, "MinimalMapReduce");
                        job.setJarByClass(MinimalMapReduce.class);
//                      设置输入输出路径
                        FileInputFormat.addInputPath(job, new Path(args[0]));
                        FileOutputFormat.setOutputPath(job, new Path(args[1]));
//                      提交作业运行
                        System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

```
yarn@host1:~/own$ hdfs dfs -cat someWords.txt
hello java
hello hadoop
hello hdfs
hello mapreduece
hello yarn
```

输入

结果

```
yarn@host1:~/own$ hdfs dfs -cat /user/yarn/MinimalMROut/part-r-00000
0       hello java
11      hello hadoop
24      hello hdfs
35      hello mapreduece
52      hello yarn
63
```

默认的MapReduce程序

- 默认的InputFormat
  - org.apache.hadoop.mapreduce.lib.input.TextInputFormat
- 默认的OutputFormat
  - org.apache.hadoop.mapreduce.lib.output.TextOutputFormat
- 默认的Mapper
  - org.apache.hadoop.mapreduce.Mapper　　//将key-value原样输出
- 默认的Reducer
  - org.apache.hadoop.mapreduce.Reducer　　//将key-value原样输出
- 默认的Reducer数量为1
- 默认的Partitioner
  - org.apache.hadoop.mapreduce.lib.partition.HashPartitioner
- 默认的输出key类型
  - org.apache.hadoop.io.LongWritable
- 默认的输出value类型
  - org.apache.hadoop.io.Text

No Reducer的MapReduce程序--Mapper

```
/**
 * 将输入的文本内容拆分为word,做一个简单输出的Mapper
 */
public class TokenCounterMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
		private Text word=new Text();
		private static final IntWritable one=new IntWritable(1);

		@Override
		protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context)
				throws IOException, InterruptedException {
			// TODO Auto-generated method stub
			StringTokenizer itr=new StringTokenizer(value.toString());

			while(itr.hasMoreTokens()){
				word.set(itr.nextToken());
				context.write(word, one);
			}

		}
}
```

No Reducer的MapReduce程序--Driver

```
/**
 *没有设置Reducer的MR程序
 */
public class NoReducerMRDriver {
        public static void main(String[] args) throws Exception {
//              构建新的作业
                Configuration conf=new Configuration();
                Job job = Job.getInstance(conf, "NoReducer");
                job.setJarByClass(NoReducerMRDriver.class);
//              设置Mapper
                job.setMapperClass(TokenCounterMapper.class);
//              设置reducer的数量为0
                job.setNumReduceTasks(0);
//              设置输出格式
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(IntWritable.class);
//              设置输入输出路径
                FileInputFormat.setInputPaths(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
//              提交运行作业
                System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

No Reducer的MapReduce程序—结果

```
yarn@host1:~/own$ hdfs dfs -cat someWords.txt
hello java
hello hadoop
hello hdfs
hello mapreduece
hello yarn
```

输入

结果

```
yarn@host1:~/own$ hdfs dfs -cat /user/yarn/NoReducerMROut_1/part-m-00000
hello   1
java    1
hello   1
hadoop  1
hello   1
hdfs    1
hello   1
mapreduece      1
hello   1
yarn    1
yarn@host1:~/own$
```

- 如果作业拥有0个Reducer,则Mapper结果直接写入OutputFormat而不经key值排序

No Mapper的MapReduce程序--Reducer

```
/**
 * 将reduce输入的values内容拆分为word,做一个简单输出的Reducer
 */
public class TokenCounterReducer extends Reducer<LongWritable, Text, Text, IntWritable>{
	private Text word=new Text();
	private static final IntWritable one=new IntWritable(1);
	@Override
	protected void reduce(LongWritable key, Iterable<Text> values,Reducer<LongWritable, Text, Text, IntWritable>.Context context)
			throws IOException, InterruptedException {
			// TODO Auto-generated method stub
			for(Text value:values){
					StringTokenizer itr=new StringTokenizer(value.toString());

					while(itr.hasMoreTokens()){
							word.set(itr.nextToken());
							context.write(word, one);
					}
			}

	}
}
```

No Mapper的MapReduce程序--Driver

```
/**
 *没有设置Mapper的MR程序
 */
public class NoMapperMRDriver {
        public static void main(String[] args) throws Exception {
//              构建新的作业
                Configuration conf=new Configuration();
                Job job = Job.getInstance(conf, "NoMapper");
                job.setJarByClass(NoMapperMRDriver.class);
//              设置Reducer
                job.setReducerClass(TokenCounterReducer.class);
//              设置输出格式
                job.setMapOutputKeyClass(LongWritable.class);
                job.setMapOutputValueClass(Text.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(IntWritable.class);
//              设置输入输出路径
                FileInputFormat.setInputPaths(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
//              提交运行作业
                System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

No Mapper的MapReduce程序—结果

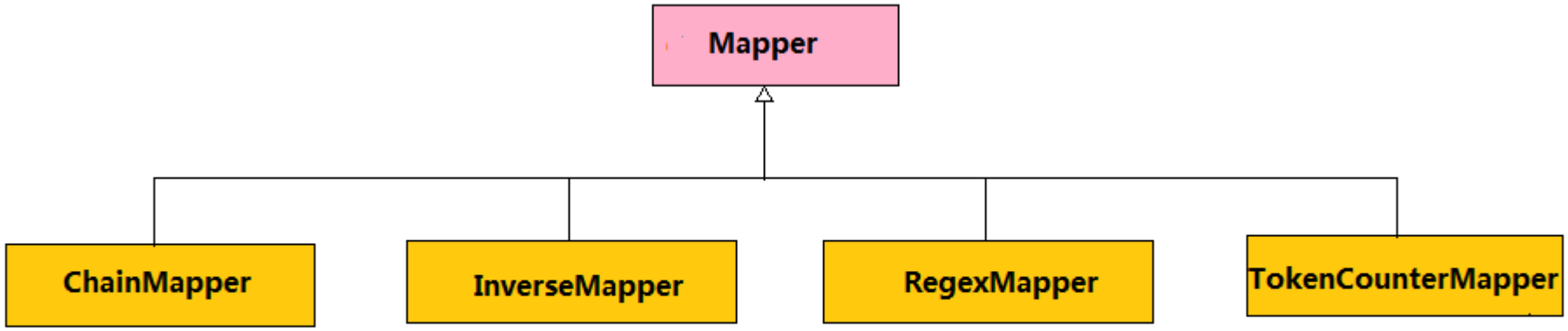
```
yarn@host1:~/own$ hdfs dfs -cat someWords.txt
hello java
hello hadoop
hello hdfs
hello mapreduece
hello yarn
```

输入

结果

```
yarn@host1:~/own$ hdfs dfs -cat /user/yarn/NoMapperMROut/part-r-00000
hello   1
java    1
hello   1
hadoop  1
hello   1
hdfs    1
hello   1
mapreduece      1
hello   1
yarn    1
```

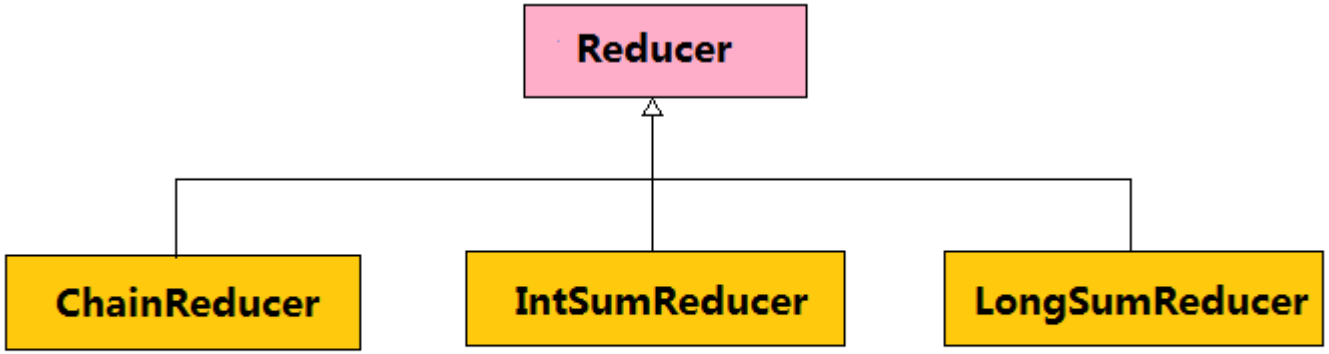
- Mapper:封装了应用程序Mapper阶段的数据处理逻辑



Hadoop MapReduce自带Mapper实现的类层次图

- ChainMapper:方便用户编写链式Map任务，即Map阶段包含多个Mapper
- InverseMapper:一个能交换key和value的Mapper
- RegexMapper:检查输入是否匹配某正则表达式，输出匹配字符串和计数器（1）
- TockenCounterMapper:将输入分解为独立的单词，输出个单词和计数器（1）

- Reducer:封装了应用程序Reducer阶段的数据处理逻辑

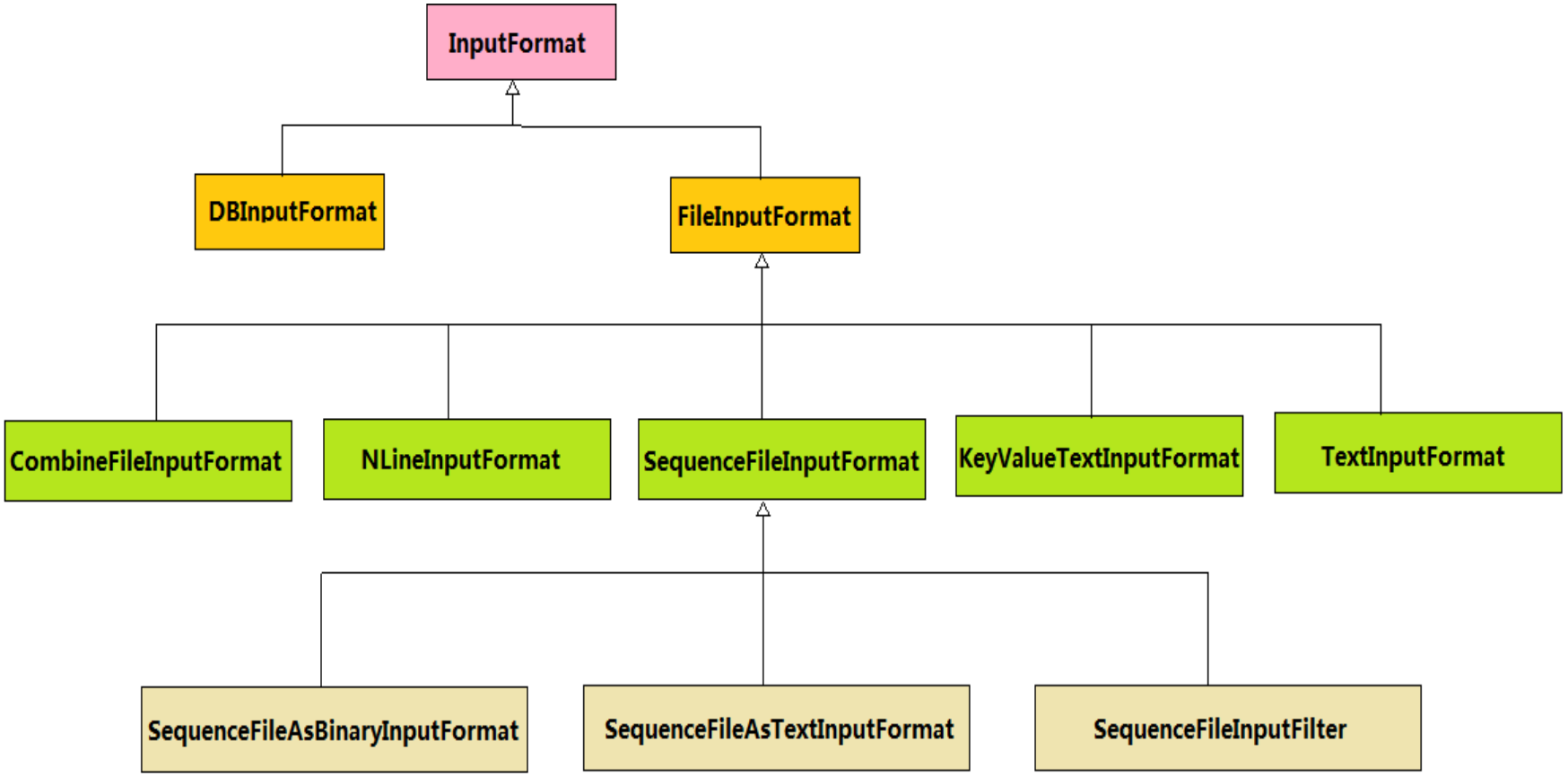


Hadoop MapReduce自带Reducer实现的类层次图

- ChainReducer:方便用户编写链式Reduce任务，即Reduce阶段包含多个Reducer
- IntSumReducer/LongSumReducer:对各key的所有整型值求和

使用InputFormat控制MapReduce执行

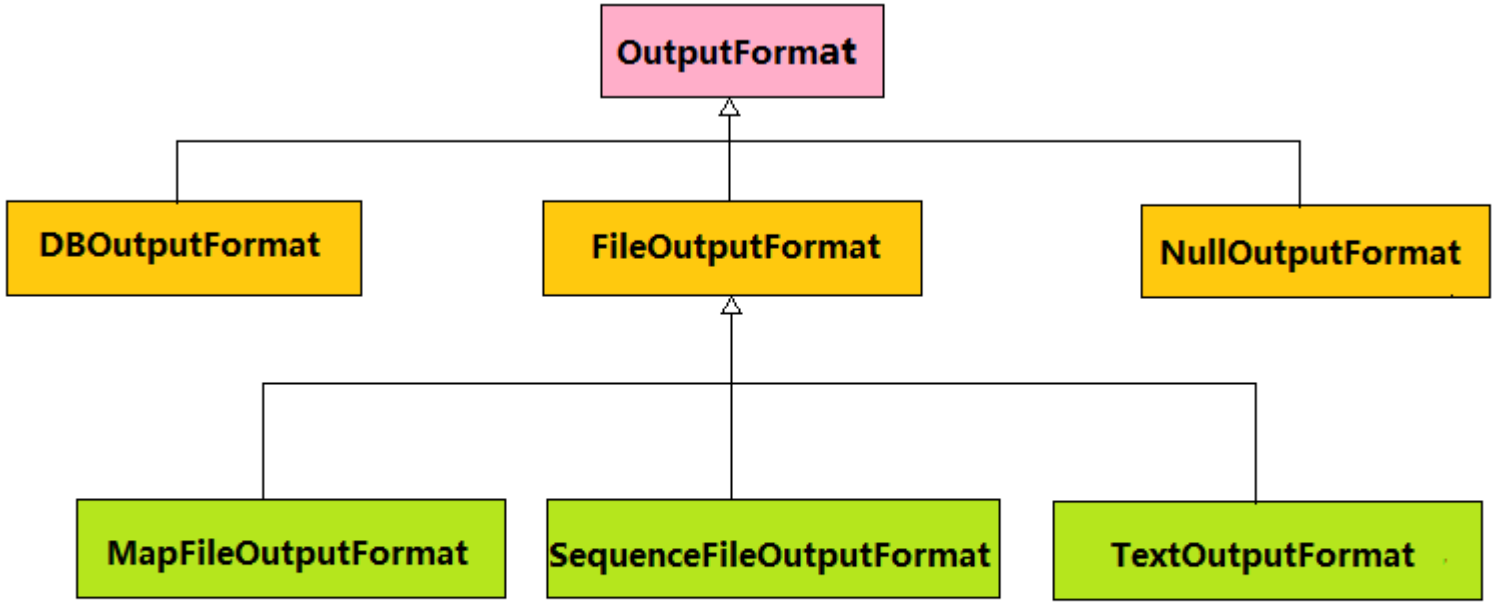
- InputFormat:描述输入数据的格式，主要完成数据切分(InputSplit)和为Mapper提供输入数据(RecordReader)两个功能



Hadoop MapReduce自带InputFormat实现的类层次图

使用OutputFormat控制MapReduce执行

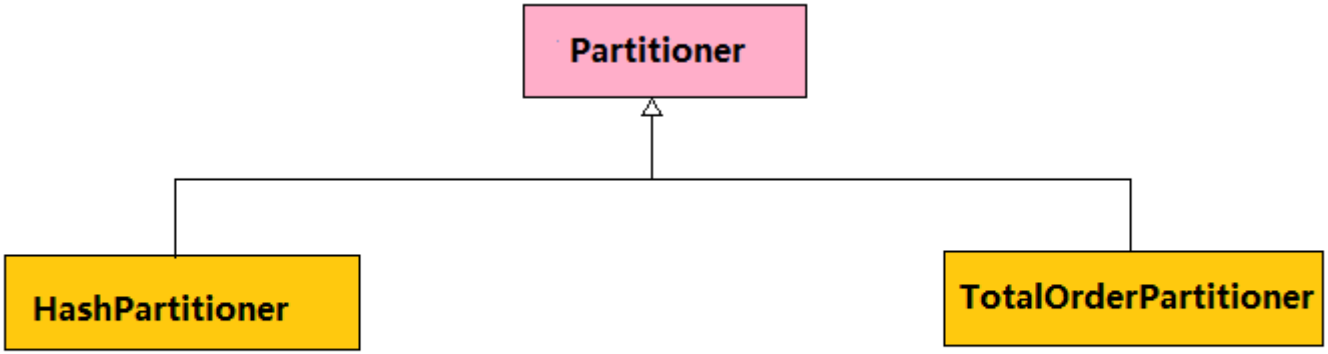
- OutputFormat:描述输出数据的格式，能够将用户提供的Key/Value对写入指定格式的文件中



Hadoop MapReduce自带OutputFormat实现的类层次图

使用Partitioner控制Reduce执行

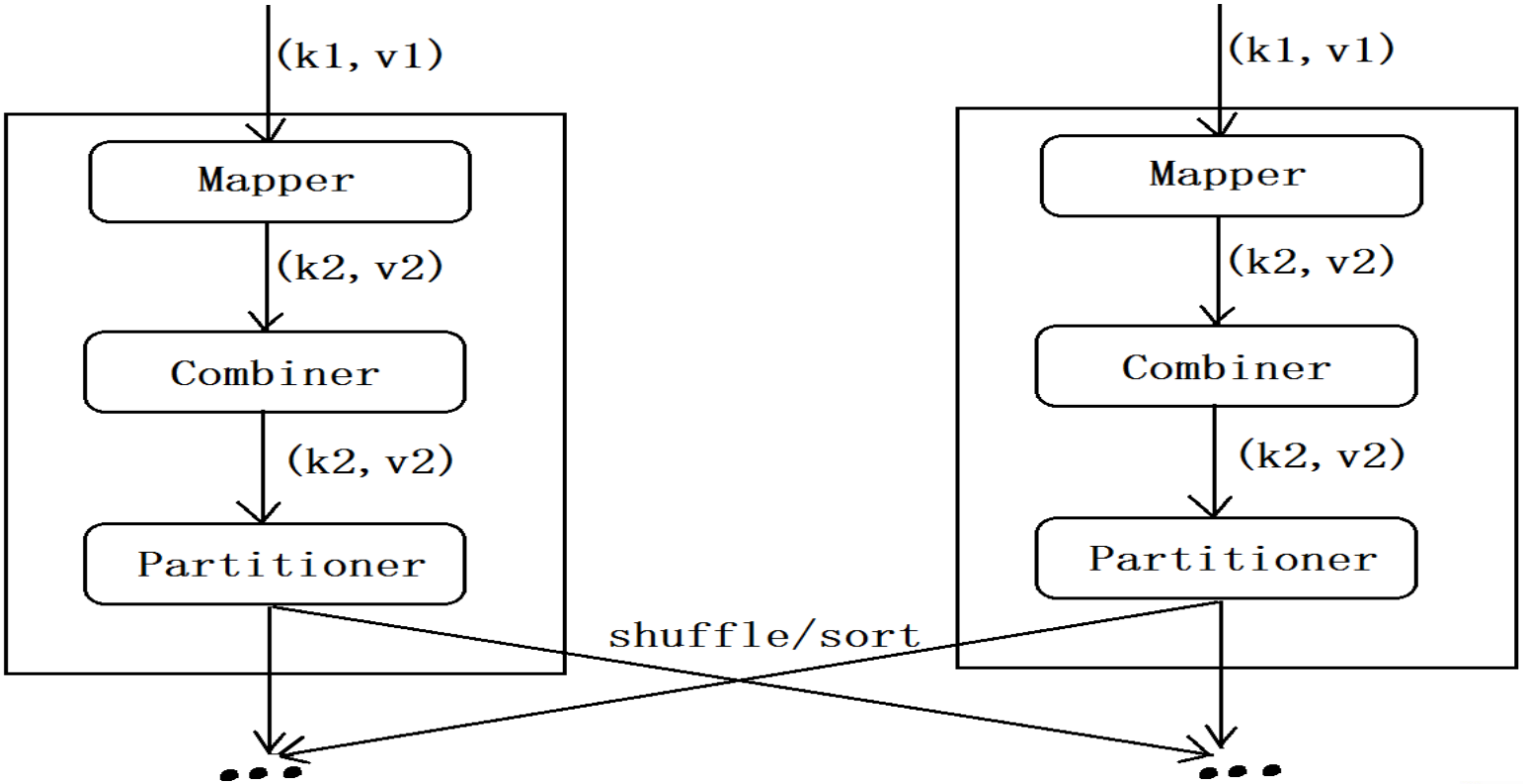
- Partitioner:对Mapper产生的中间结果进行分片，以便将同一分组的数据交给同一Reducer处理



Hadoop MapReduce自带Partitioner实现类的层次图

使用Combiner优化MapReduce执行

- 前提：每一个map都可能会产生大量的本地输出
- Combiner功能：对map端的输出先做一次合并
- 目的：减少在map和reduce节点之间的数据传输量，以提高网络IO性能



Hadoop工作流

- JobControl：方便用户编写有依赖关系的作业，这些作业往往构成一个有向图。
- Hadoop工作流引擎

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop全分布式环境搭建

Brighten Your Way And Raise You Up.

1. 前期准备

- Linux
- JDK
- hadoop-2.6.0

2. 规划集群结构

假定有如下3台机器，名字分别为hadoop1, hadoop2, hadoop3，所有机器都有用户briup

- 功能规划如下：
  - hadoop1: 集群主节点，运行NameNode和ResourceManager进程
  - hadoop2: 运行SecondaryNameNode、DataNode和NodeManager进程
  - hadoop3:运行jobhistoryserver、DataNode、NodeManager继承
- IP地址规划：
  - 192.168.150.128　　hadoop1
  - 192.168.150.129　　hadoop2
  - 192.168.150.130　　hadoop3
- 将IP地址和主机名对应关系写入到每台机器的/etc/hosts文件中

3. 配置主机间briup用户ssh无密码登录

- 以其他主机免密登录hadoop1为例，基本步骤
  - 在hadoop1机器上生成密钥对(密码为空字符串)：
    - hadoop1$> ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
  - 将hadoop1机器上的~/.ssh/id_rsa.pub复制到其他机器上
    - hadoop1$>scp ~/.ssh/id_rsa.pub briup@hadoop2:~
    - hadoop1$> scp ~/.ssh/id_rsa.pub briup@hadoop3:~
  - 将上述公钥各自的~/.ssh/authorized_keys文件中
    - hadoop1$>cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
    - hadoop1$> ssh briup@hadoop2
    - hadoop2$> cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
    - hadoop2$> ssh briup@hadoop3
    - hadoop3$> cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
  - 修改所有机器目录权限如下:
    - chmod 700 ~/.ssh
    - chmod 600 ~/.ssh/authorized_keys
- 其他主机类似进行配置

4. 每台主机安装Hadoop

- 解压下载好的hadoop-x.x.x.tar.gz
  - tar -zvxf hadoop-2.6.0.tar.gz
- 编辑hadoop-2.6.0/etc/hadoop/hadoop-env.sh
  - export JAVA_HOME=/opt/jdk1.7.0_79
  - export HADOOP_PREFIX=/opt/hadoop-2.6.0
- 在.bashrc下配置PATH
  - PATH=...../opt/hadoop-2.6.0/bin:/opt/hadoop-2.6.0/sbin
- 验证是否安装成功
  - hadoop version

5.1全分布模式配置之core-site.xml

- core-site.xml

```
<configuration>
        <!--指定Hadoop所用的文件系统-->
        <property>
                <name>fs.defaultFS</name>
                <value>hdfs://192.168.150.128:9000</value>
        </property>
        <!--指定Hadoop运行时产生文件的存储目录-->
        <property>
                <name>hadoop.tmp.dir</name>
                <value>file:/opt/hadoop-2.6.0/tmp</value>
        </property>
</configuration>
```

5.2全分布模式配置之hdfs-site.xml

- hdfs-site.xml

```
<configuration>
        <!--指定HDFS副本的数量-->
        <property>
                <name>dfs.replication</name>
                <value>3</value>
        </property>

        <property>
                <name>dfs.namenode.name.dir</name>
                <value>file:/opt/hadoop_data/hdfs/nn</value>
        </property>

        <property>
                <name>fs.checkpoint.dir</name>
                <value>file:/opt/hadoop_data/hdfs/snn</value>
        </property>

        <property>
                <name>fs.checkpoint.edits.dir</name>
                <value>file:/opt/hadoop_data/hdfs/snn</value>
        </property>

        <property>
                <name>dfs.datanode.data.dir</name>
                <value>file:/opt/hadoop_data/hdfs/dn</value>
        </property>
        <property>
                <name>dfs.nameservices</name>
                <value>cluster_1</value>
        </property>
        <property>
                <name>dfs.namenode.secondary.http-address</name>
                <value>192.168.150.129:50090</value>
        </property>

</configuration>
```

5.3全分布模式配置之mapred-site.xml

- mapred-site.xml

```
<configuration>
        <!-- 指定MapReduce程序运行在Yarn上-->
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
</configuration>
```

5.4全分布模式配置之yarn-site.xml

- yarn-site.xml

```
<configuration>
        <!--指定ResourceManager的地址-->
        <property>
                <name>yarn.resourcemanager.address</name>
                <value>192.168.150.128:18040</value>
        </property>
        <property>
                <name>yarn.resourcemanager.scheduler.address</name>
                <value>192.168.150.128:18030</value>
        </property>
        <property>
                <name>yarn.resourcemanager.resource-tracker.address</name>
                <value>192.168.150.128:18025</value>
        </property>
        <!--指定reducer获取数据的方式-->
        <property>
                <name>yarn.nodemanager.aux-services</name>
                <value>mapreduce_shuffle</value>
        </property>
        <property>
                <name>yarn.resourcemanager.admin.address</name>
                <value>192.168.150.128:18041</value>
        </property>
</configuration>
~
~
~
```

6. 在每台主机创建HDFS系统相关的目录及修改相关权限

- sudo mkdir -p /opt/hadoop_data/hdfs/nn
- sudo mkdir -p /opt/hadoop_data/hdfs/dn
- sudo mkdir -p /opt/hadoop_data/hdfs/snn
- sudo chown -R briup /opt/hadoop_data/hdfs
- sudo chown –R briup /opt/hadoop-2.6.0

7. 在每台主机格式化HDFS文件系统

- hdfs namenode –format

8.1在相应主机启动HDFS相关进程

- HDFS集群启动
  - Hadoop1:
    - hadoop-daemon.sh start namenode
  - Hadoop2:
    - hadoop-daemon.sh start datanode
    - hadoop-daemon.sh start secondarynamenode
  - Hadoop3:
    - hadoop-daemon.sh start datanode
  - 监控平台：
    - http://hadoop1:50070

8.2在相应主机启动YARN和MapReduce相关进程

- YARN集群启动
  - Hadoop1:
    - yarn-daemon.sh start redourcemanager
  - Hadoop2:
    - yarn-daemon.sh start nodemanager
  - Hadoop3:
    - yarn-daemon.sh start nodemanager
    - mr-jobhistory-daemon.sh start historyserver
  - 监控平台：
    - http://hadoop1:8088

9.1在相应主机关闭HDFS相关进程

- HDFS集群关闭
  - Hadoop1:
    - hadoop-daemon.sh stop namenode
  - Hadoop2:
    - hadoop-daemon.sh stop datanode
    - hadoop-daemon.sh stop secondarynamenode
  - Hadoop3:
    - hadoop-daemon.sh stop datanode

9.2在相应主机启动YARN和MapReduce相关进程

- YARN集群启动
  - Hadoop1:
    - yarn-daemon.sh stop redourcemanager
  - Hadoop2:
    - yarn-daemon.sh stop nodemanager
  - Hadoop3:
    - yarn-daemon.sh stop nodemanager
    - mr-jobhistory-daemon.sh stop historyserver

杰普软件科技有限公司
www.briup.com

公司总部：上海市闸北区万荣路1188弄龙软软件园区A栋206室
电话：（021）56778147
邮政编码：200436

昆山实训基地：昆山市巴城学院路828号昆山浦东软件园北楼4-5层
电话：（0512）50190290-8000
邮政编码：215311

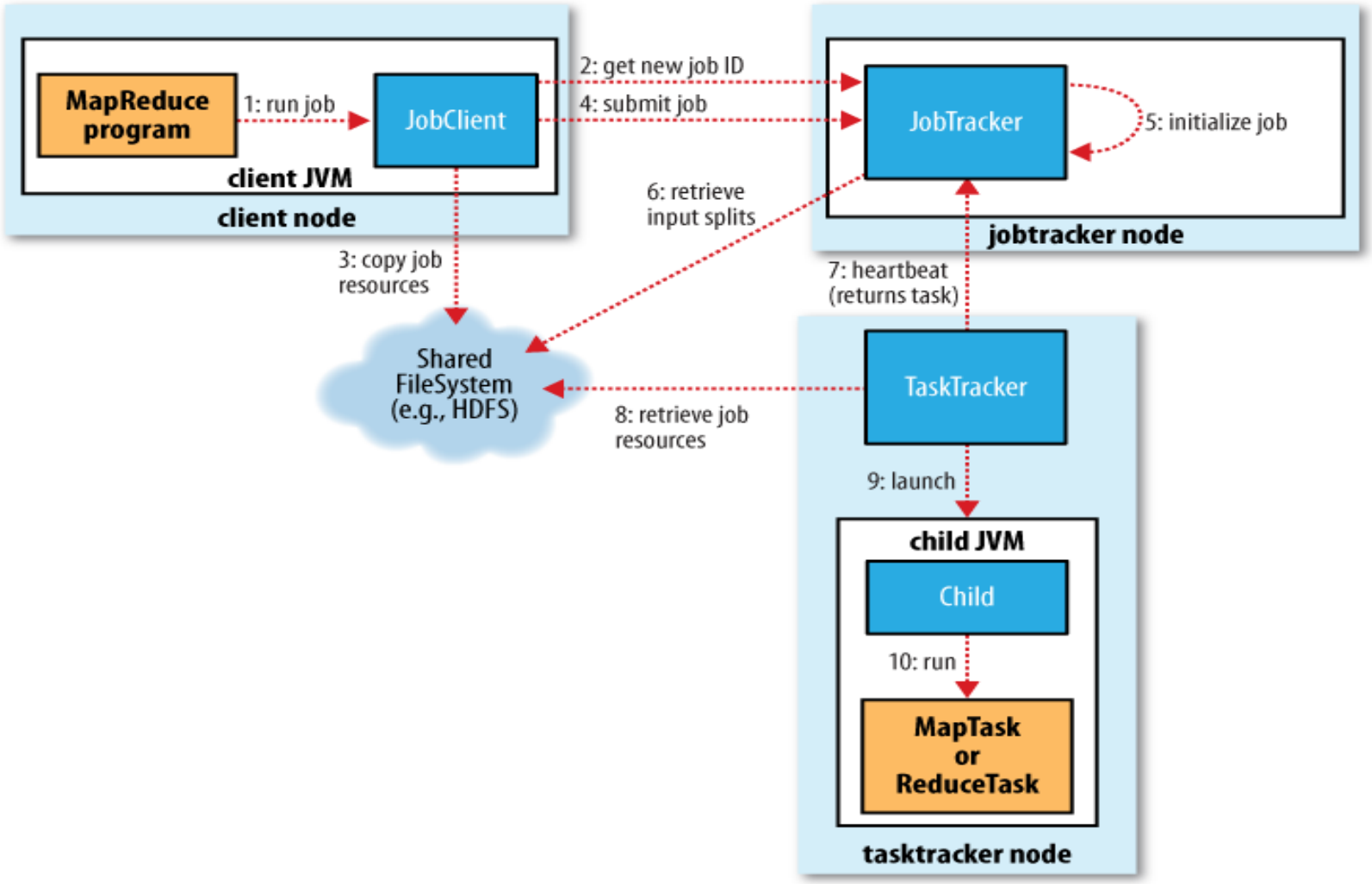
电邮：training@briup.com
主页：http://www.briup.com

Briup High-End IT Training

Hadoop之Yarn

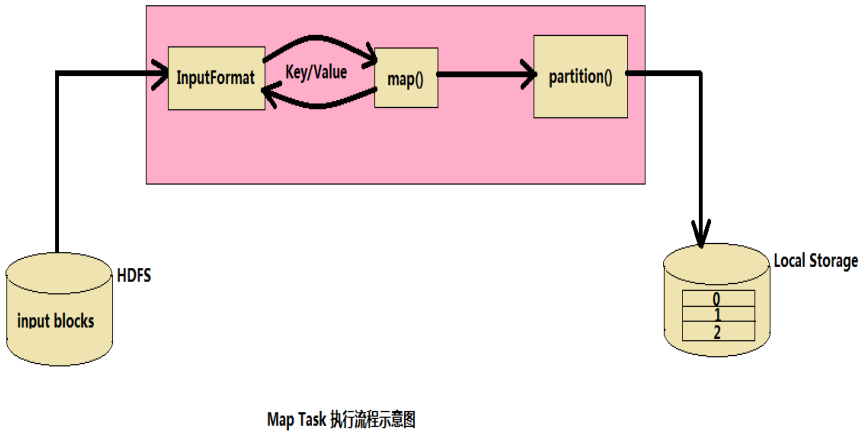
Brighten Your Way And Raise You Up.

- Yet Another Resource Negotiator，另一种资源协调者
- 为上层应用提供统一的资源管理和调度
- 解决关于集群利用率、资源统一管理和数据共享等问题

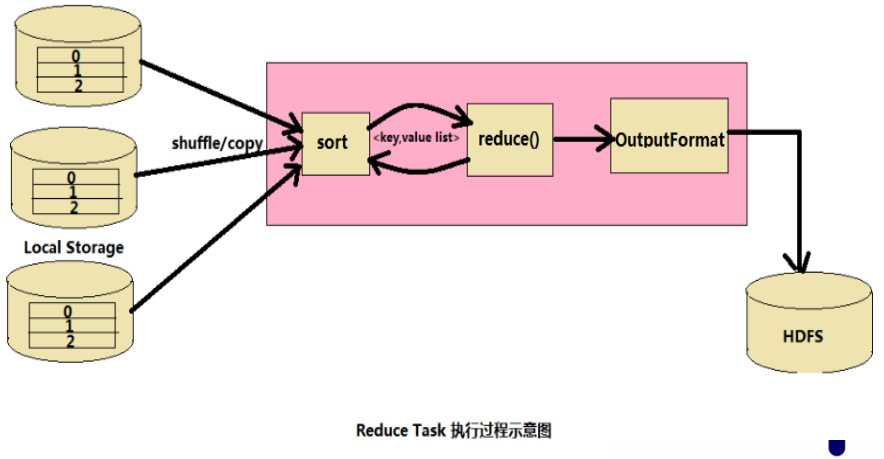

MapReduce program
1: run job
JobClient
client JVM
client node
2: get new job ID
4: submit job
JobTracker
5: initialize job
jobtracker node
6: retrieve input splits
3: copy job resources
7: heartbeat (returns task)
Shared FileSystem (e.g., HDFS)
8: retrieve job resources
TaskTracker
9: launch
child JVM
Child
10: run
MapTask or ReduceTask
tasktracker node

MRv1的核心组件

- JobTracker
  - 主要负责资源监控和作业调度。
- TaskTracker
  - 周期性地通过Heartbeat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）
- Task
  - 分为Map Task和Reduce Task两种，均由TaskTracker启动。



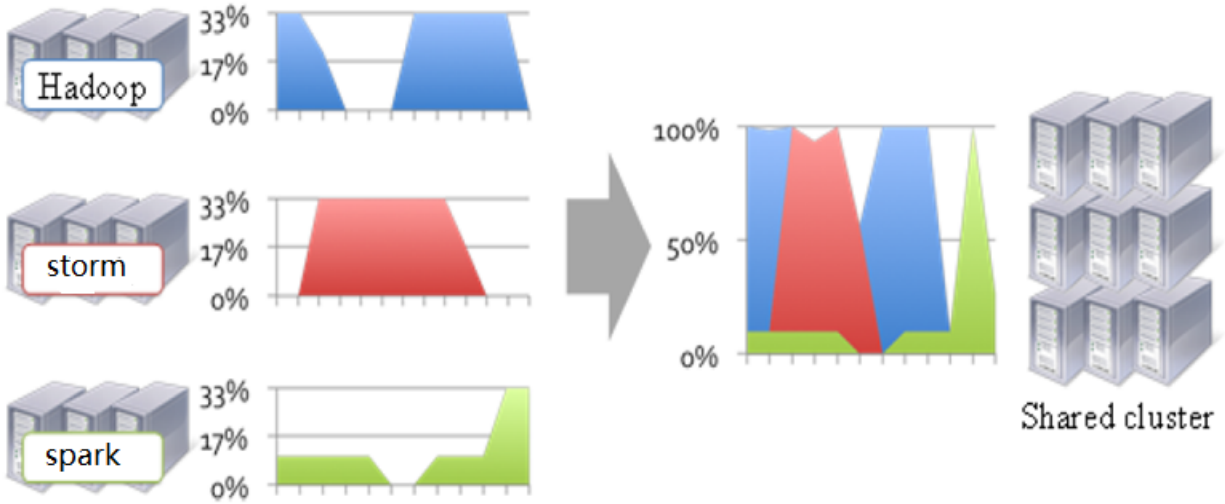
Map Task 执行流程示意图



Reduce Task 执行过程示意图

MRv1的缺陷

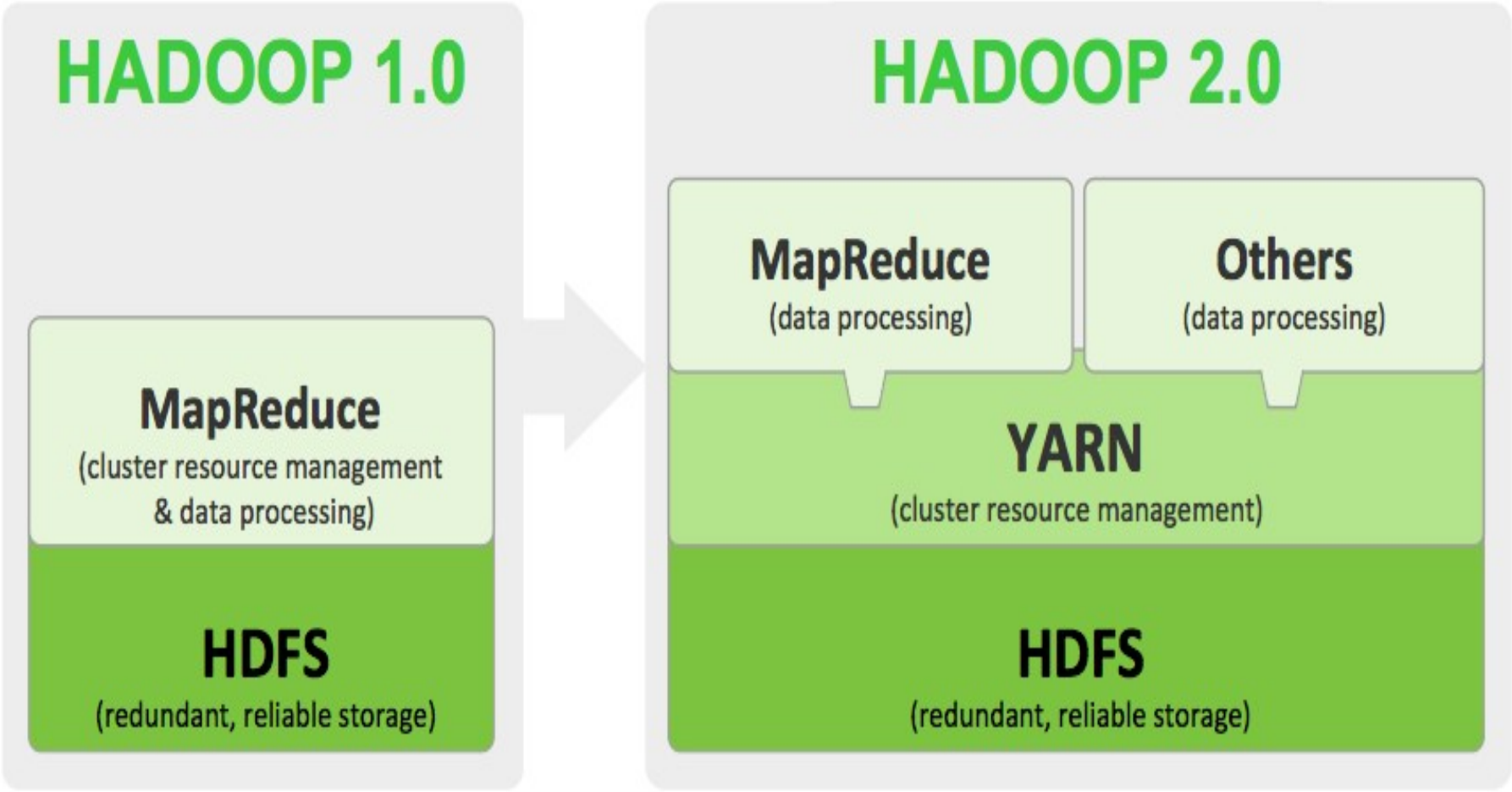
- 扩展性受限
- 单点故障
- 难以支持MR之外的计算框架



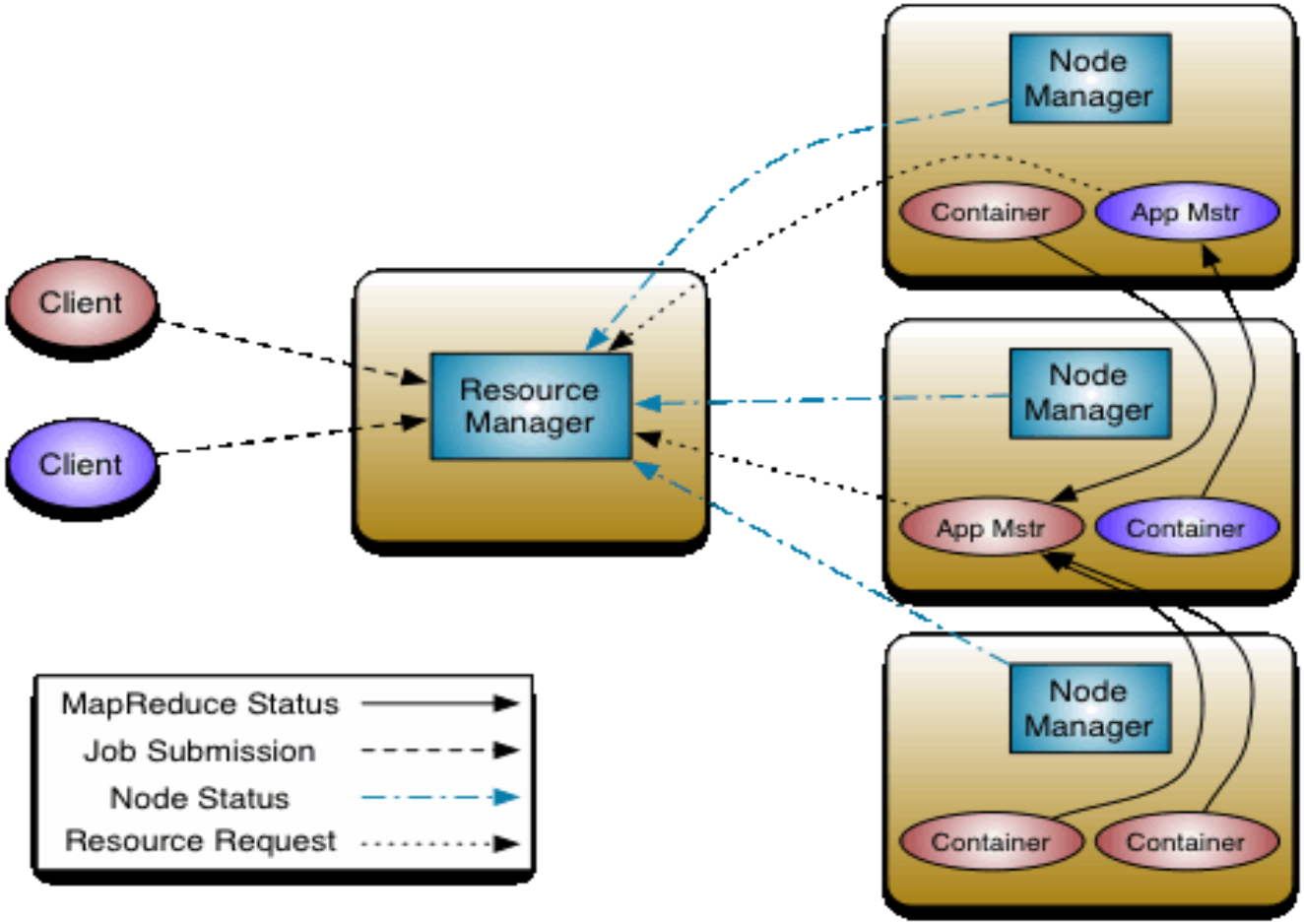
多种计算框架的出现

多计算框架各自为战，数据共享困难

- MR：离线计算框架
- Storm：实时计算框架
- Spark：内存计算框架

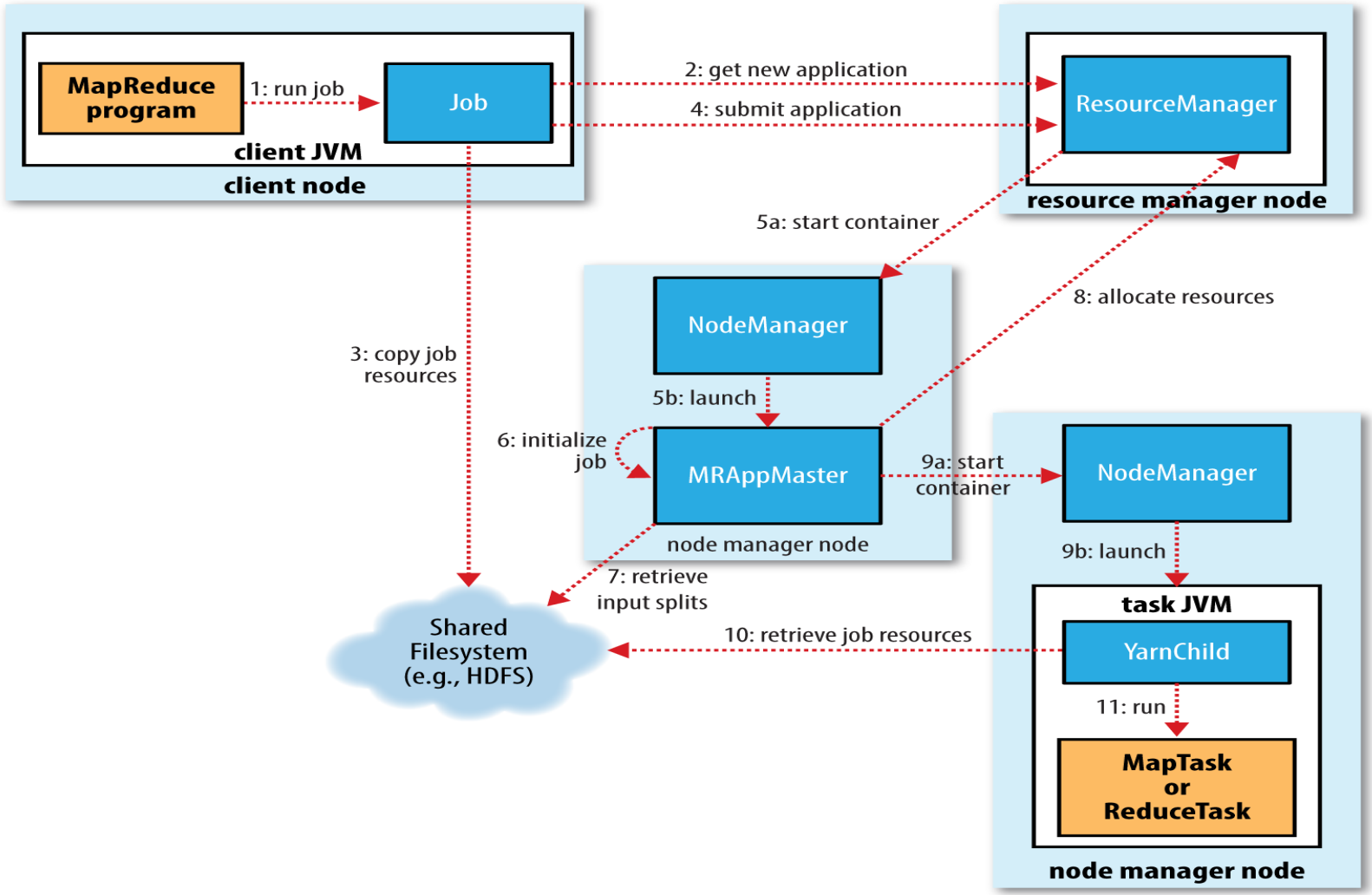

HADOOP 1.0
MapReduce
(cluster resource management
& data processing)
HDFS
(redundant, reliable storage)
HADOOP 2.0
MapReduce
(data processing)
Others
(data processing)
YARN
(cluster resource management)
HDFS
(redundant, reliable storage)

YARN的体系结构


Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

YARN的核心组件

- ResourceManager(RM)：整个集群只有一个，负责集群资源的统一管理和调度
  - 处理客户端请求
  - 启动/监控ApplicationMaster
  - 监控NodeManager
  - 资源分配与调度
- ApplicationMaster(AM)：每个应用有一个，负责应用程序的管理
  - 数据切分
  - 为应用程序申请资源，并分配给内部任务
  - 任务监控与容错
- NodeManager(NM)：整个集群有多个，负责单节点资源管理和使用
  - 单个节点上的资源管理
  - 处理来自ResourceManager的命令
  - 处理来自ApplicationMaster的命令
- Container
  - YARN中的资源分配单位，封装了多维度的资源，如内存、CPU、磁盘、网路等，目前仅支持CPU和内存两种资源


MapReduce program
1: run job
Job
client JVM
client node
2: get new application
4: submit application
ResourceManager
resource manager node
5a: start container
8: allocate resources
NodeManager
5b: launch
3: copy job resources
6: initialize job
MRAppMaster
node manager node
9a: start container
NodeManager
9b: launch
task JVM
YarnChild
11: run
MapTask or ReduceTask
node manager node
7: retrieve input splits
Shared Filesystem (e.g., HDFS)
10: retrieve job resources