法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象学院
  - 新浪微博：小象AI学院

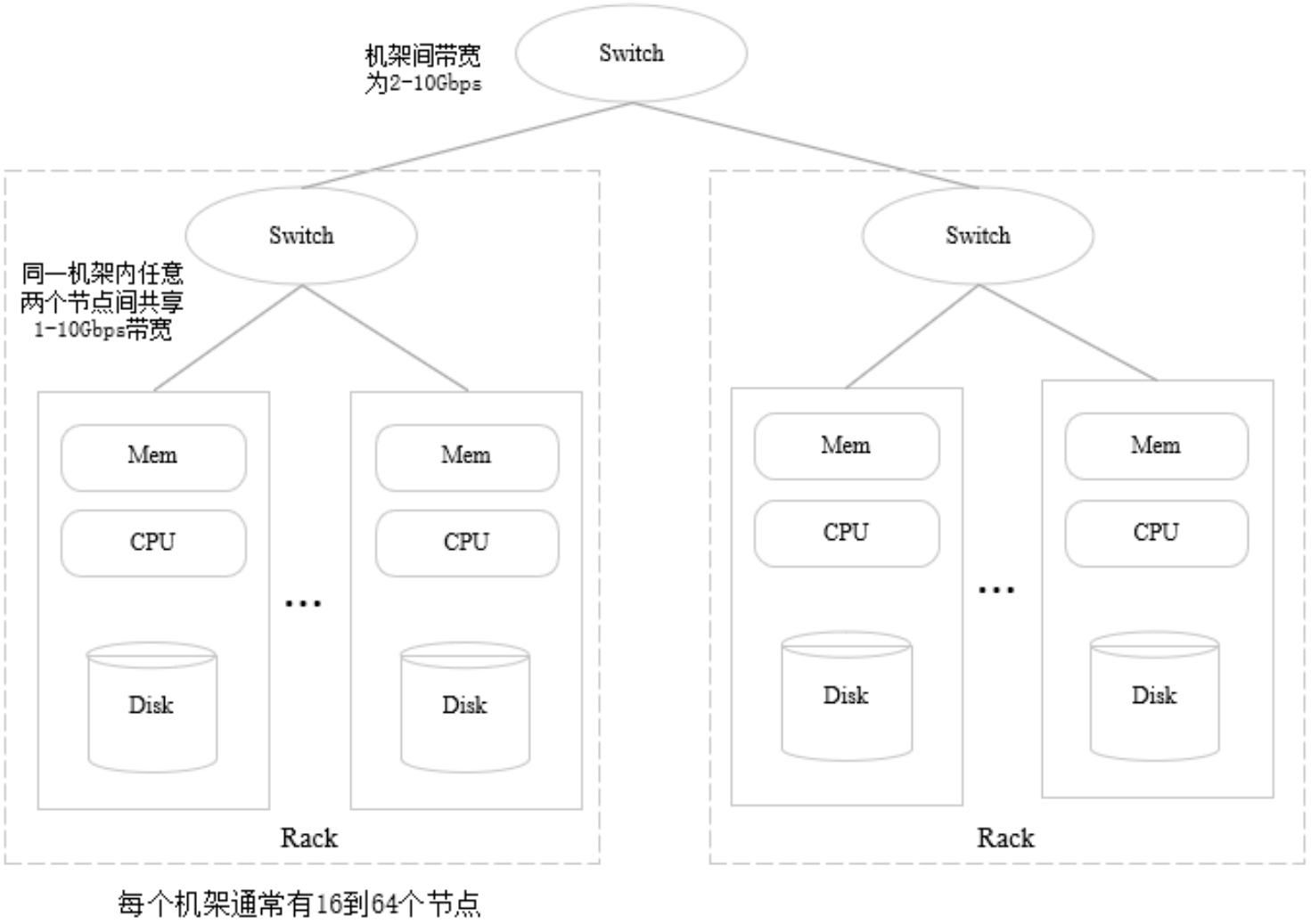
大纲

- HDFS内部机制
- HDFS文件操作与管理
- YARN作业运行过程
- MapReduce原理与特性
- MapReduce编程模型
- 编程接口

大纲

HDFS内部机制

物理拓扑


机架间带宽
为2-10Gbps
Switch
Switch
Switch
同一机架内任意
两个节点间共享
1-10Gbps带宽
Mem
CPU
Disk
Mem
CPU
Disk
Rack
Mem
CPU
Disk
Mem
CPU
Disk
Rack
每个机架通常有16到64个节点

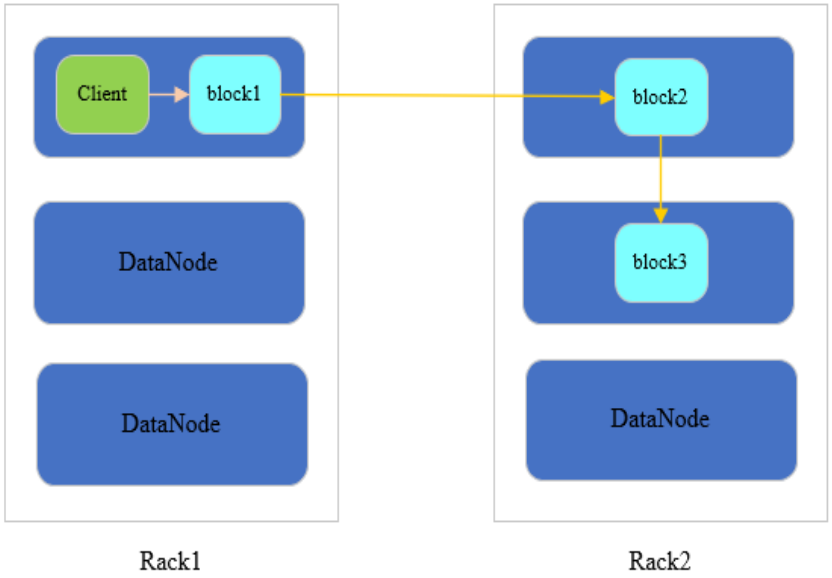
Block副本放置策略

副本1：同Client的节点上

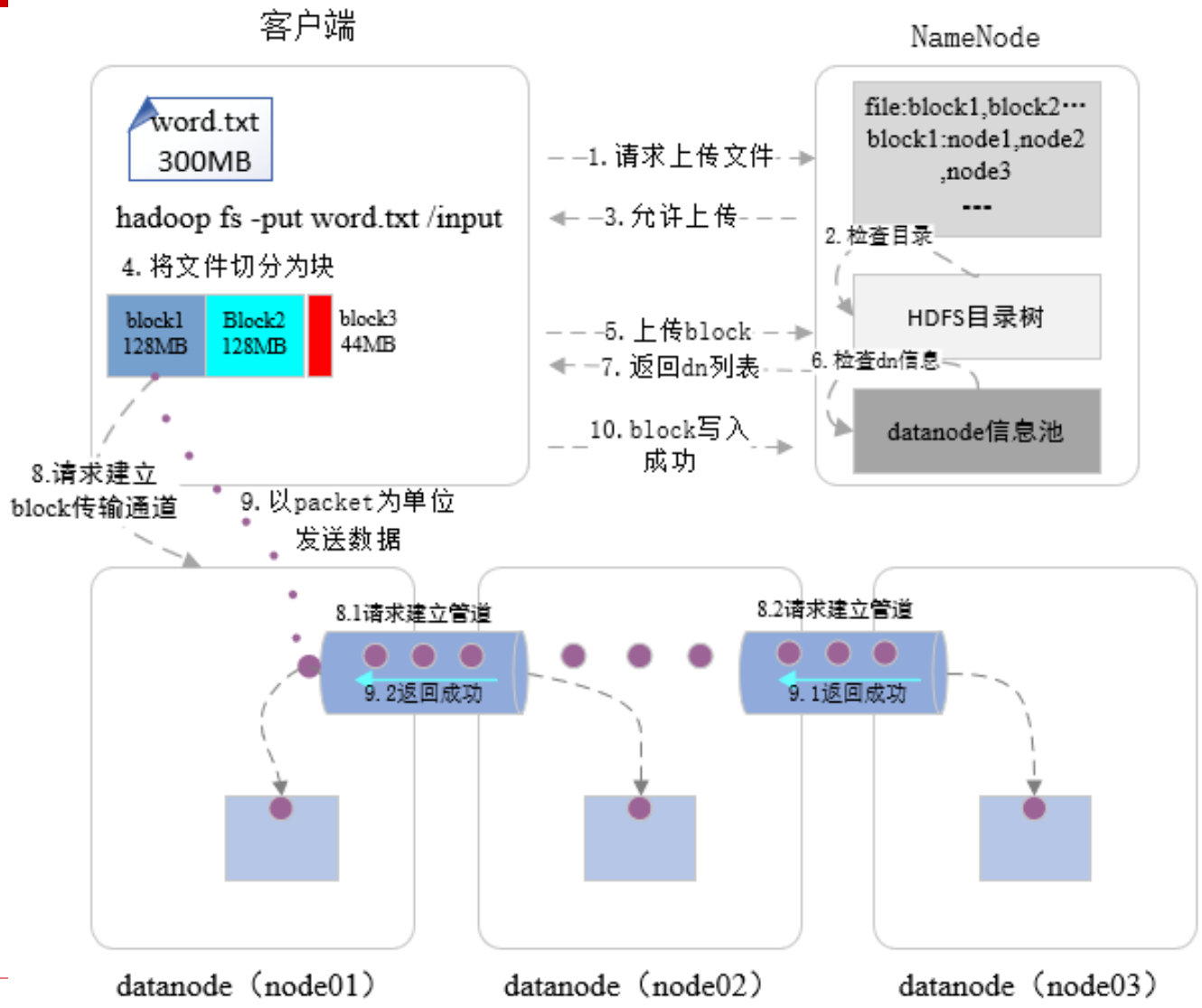
副本2：不同机架中的节点上

副本3：与第二个副本同一机架的另一个节点上

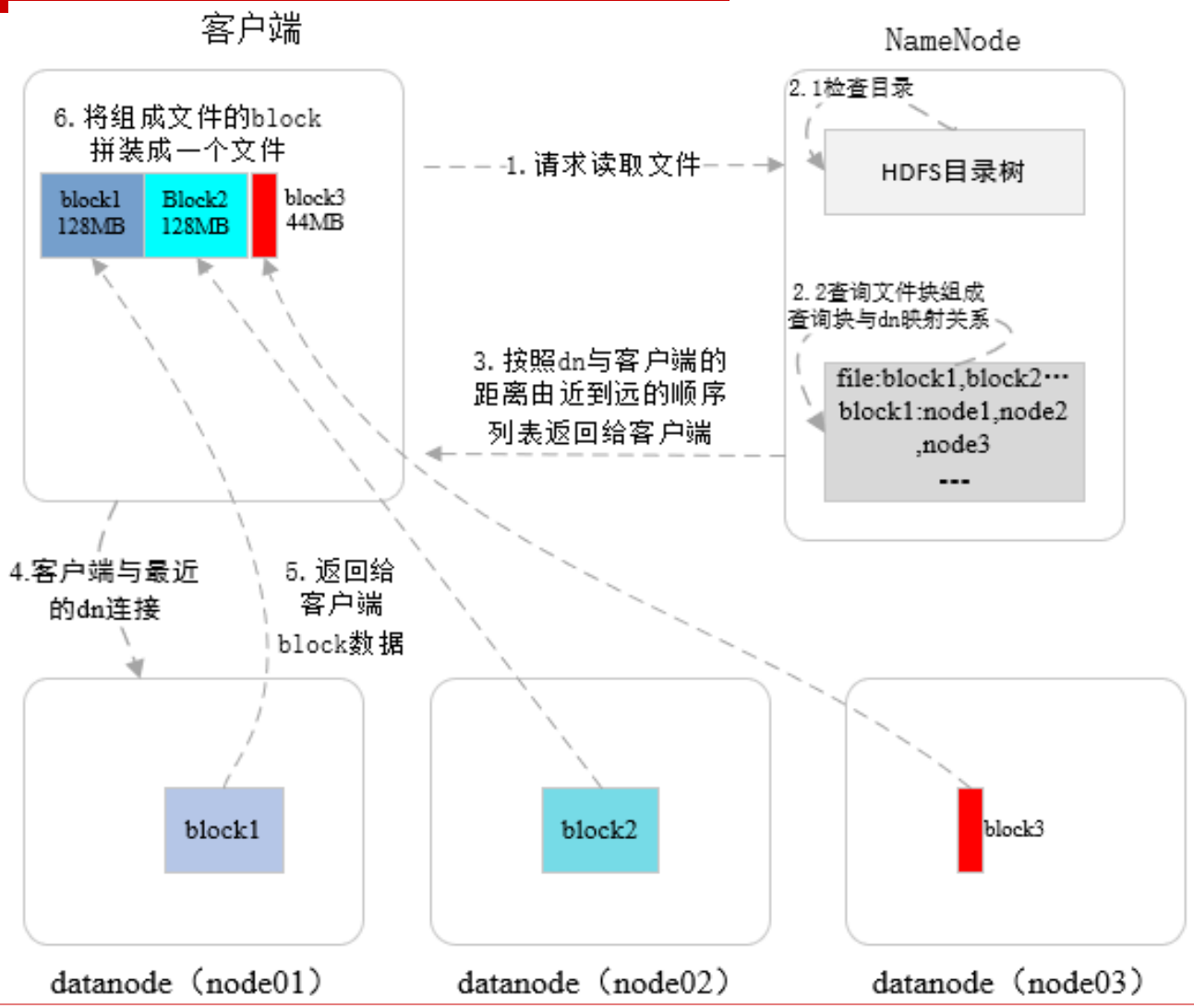
其他副本：随机挑选



HDFS文件写入流程

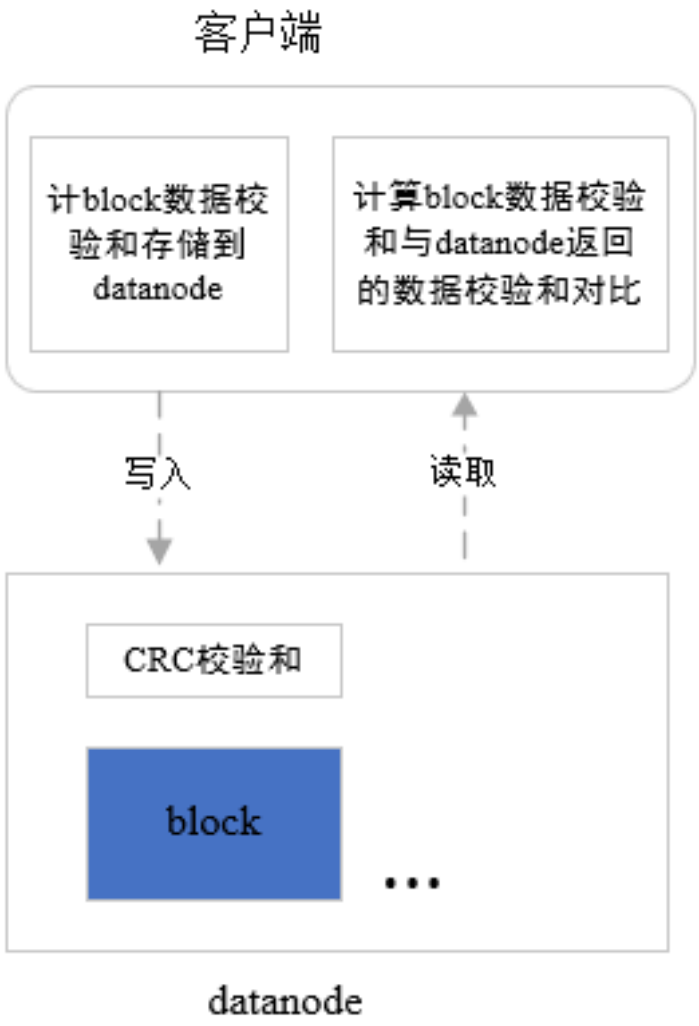

客户端
NameNode
word.txt
300MB
hadoop fs -put word.txt /input
4. 将文件切分为块
block1
128MB
Block2
128MB
block3
44MB
1. 请求上传文件
3. 允许上传
file:block1,block2…
block1:node1,node2
,node3
---
2. 检查目录
HDFS目录树
5. 上传block
7. 返回dn列表
6. 检查dn信息
datanode信息池
10. block写入
成功
8.请求建立
block传输通道
9. 以packet为单位
发送数据
8.1请求建立管道
9. 2返回成功
8.2请求建立管道
9. 1返回成功
datanode（node01）
datanode（node02）
datanode（node03）

HDFS文件读取流程


客户端
NameNode
6. 将组成文件的block
拼装成一个文件
block1
128MB
Block2
128MB
block3
44MB
1. 请求读取文件
2. 1检查目录
HDFS目录树
2. 2查询文件块组成
查询块与dn映射关系
file:block1,block2…
block1:node1,node2
,node3
---
3. 按照dn与客户端的
距离由近到远的顺序
列表返回给客户端
4.客户端与最近
的dn连接
5. 返回给
客户端
block数据
block1
block2
block3
datanode（node01）
datanode（node02）
datanode（node03）

HDFS数据完整性

- 读写数据传输完整性
  - CRC-32
- 使用其他副本替代损坏的block
  - Datanode后台线程定期检测，通过心跳定期向NameNode汇报，NameNode标记损坏的块，在其他节点拷贝新的副本



大纲

HDFS文件操作与管理

Hadoop服务脚本

sbin目录

- start-all.sh / stop-all.sh

启动/停止Hadoop集群中的全部服务（hdfs服务和yarn服务）

- start-dfs.sh/ stop-dfs.sh

只启动/关闭HDFS相关全部服务（NameNode、DataNode、JournalNode ）

- hadoop-daemon.sh start/stop namenode/datanode/journalnode

单独启动/停止单台HDFS某个服务

- start-yarn.sh/stop-yarn.sh

启动/停止YARN相关全部服务（ResourceManager、NodeManager）

- yarn-daemon.sh start/stop resourcemanager/nodemanager

单独启动/停止单台YARN某个服务

HDFS文件操作命令

bin/hadoop fs 或者bin/hfds dfs

```
[hadoop@node01 hadoop]$ hadoop fs
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]
        [-find <path> ... <expression> ...]
        [-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>]
```

HDFS文件管理命令

bin/hdfs dfsadmin

```
[hadoop@node01 hadoop]$ hdfs dfsadmin
Usage: hdfs dfsadmin
Note: Administrative commands can only be run as the HDFS superuser.
        [-report [-live] [-dead] [-decommissioning]]
        [-safemode <enter | leave | get | wait>]
        [-saveNamespace]
        [-rollEdits]
        [-restoreFailedStorage true|false|check]
        [-refreshNodes]
        [-setQuota <quota> <dirname>...<dirname>]
        [-clrQuota <dirname>...<dirname>]
        [-setSpaceQuota <quota> [-storageType <storagetype>] <dirname>...<dirname>]
        [-clrSpaceQuota [-storageType <storagetype>] <dirname>...<dirname>]
        [-finalizeUpgrade]
        [-rollingUpgrade [<query|prepare|finalize>]]
        [-refreshServiceAcl]
        [-refreshUserToGroupsMappings]
        [-refreshSuperUserGroupsConfiguration]
        [-refreshCallQueue]
        [-refresh <host:ipc_port> <key> [arg1..argn]
        [-reconfig <datanode|...> <host:ipc_port> <start|status>]
        [-printTopology]
        [-refreshNamenodes datanode_host:ipc_port]
        [-deleteBlockPool datanode_host:ipc_port blockpoolId [force]]
        [-setBalancerBandwidth <bandwidth in bytes per second>]
        [-fetchImage <local directory>]
```

HDFS增加和移除节点

- 集群中加入新的datanode方法
  - 在新的机器上拷贝一份包含配置文件的hadoop安装包
  - 单独启动datanode

    sbin/hadoop-daemon.sh start datanode
- 从集群中移除故障或者废弃的datanode
  - 将需要移除的datanode的主机名或者IP加入到NameNode的黑名单

    加入黑名单方法：

    修改NameNode的hdfs-site.xml文件，设置dfs.hosts.exclude配置的值为需要移除的datanode的主机名或者IP
  - 更新黑名单

    bin/hadoop dfsadmin –refreshNodes

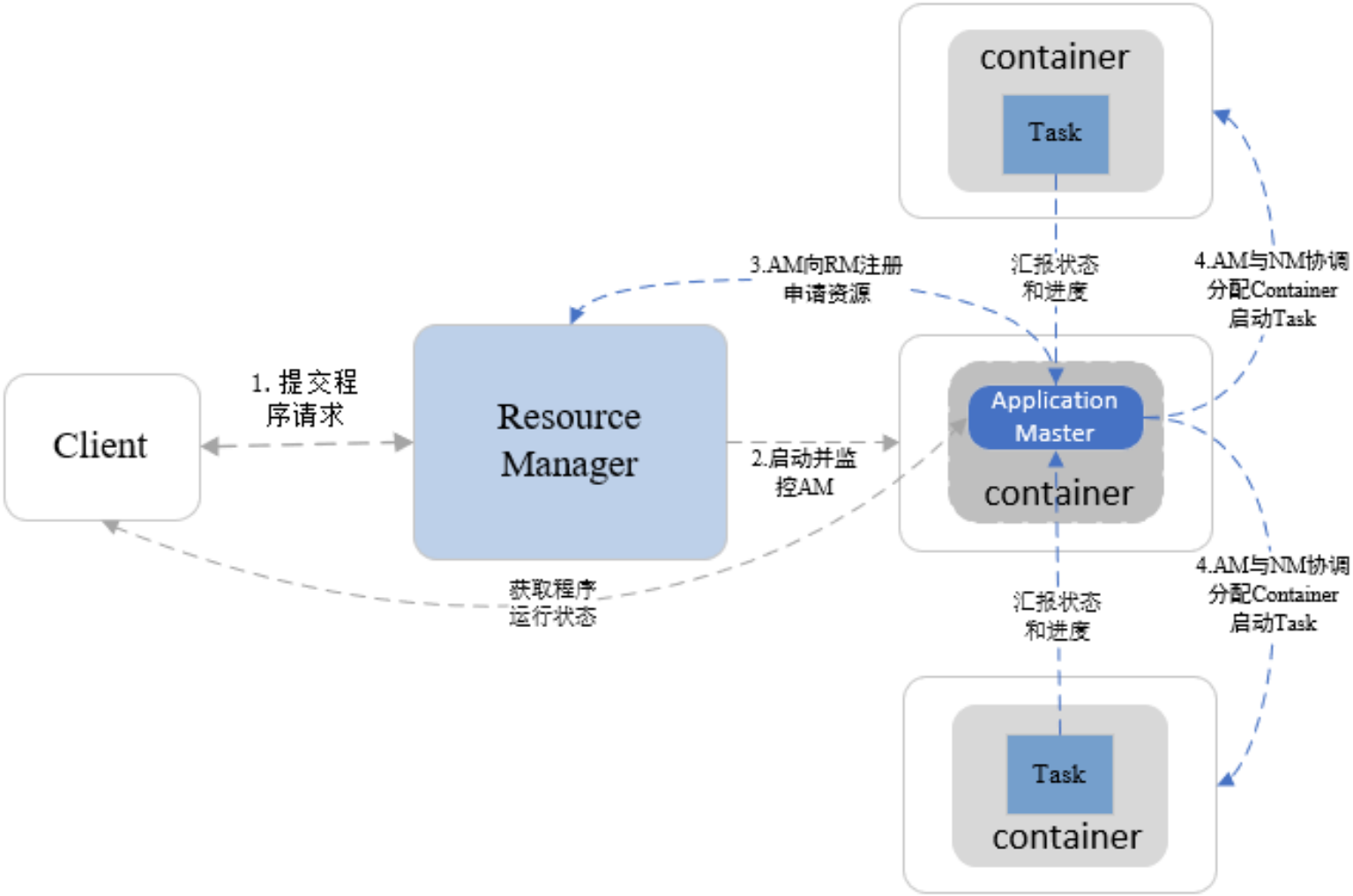
HDFS数据均衡器-balancer

- 数据块重新分布
  - sbin/start-balancer.sh -threshold 平衡阈值（默认10，表示10%）
- 平衡阈值
  - HDFS达到平衡状态的磁盘使用率偏差值
  - 值越低各节点越平衡，但消耗时间也更长

HDFS“冷” 数据处理

- “冷”数据文件
  - 很长时间没有被访问过的数据文件（如半年内）
- 处理方法
  - 高压缩比算法进行压缩，如Gzip或bzip2
  - 小文件合并

YARN程序运行流程


container
Task
3.AM向RM注册
申请资源
汇报状态
和进度
4.AM与NM协调
分配Container
启动Task
Client
1. 提交程
序请求
Resource
Manager
2.启动并监
控AM
Application
Master
container
获取程序
运行状态
4.AM与NM协调
分配Container
启动Task
汇报状态
和进度
Task
container

大纲

MapReduce原理与特性

MapReduce原理与特性

- 源自于Google的MapReduce论文
  - 发表与于2014年12月
  - Hadoop MapReduce是Goole MapReduce的克隆版
- 批处理计算框架
- 一个MapReduce程序分为Map阶段和Reduce阶段
- MapReduce特性
  - 易于编程
  - 良好的扩展性
  - 高容错性
  - 适合海量数据的离线处理

MapReduce常用应用场景

- 数据统计，比如网站的PV、UV统计
- 搜索引擎索引
- 海量数据查找
- 复杂数据分析算法实现
- 聚类算法
  - 分类算法
  - 推荐算法
  - 推荐算法

MapReduce不擅长的场景

- 实时计算
  - 能够在毫秒或者秒级内返回结果
- 流式计算
  - 输入数据集是静态的，不能够动态变化
  - 自身的设计特点决定了数据源必须是静态的
- DAG计算
  - 多个作业之间存在依赖关系，后一个应用程序输入为前一个应用程序输出

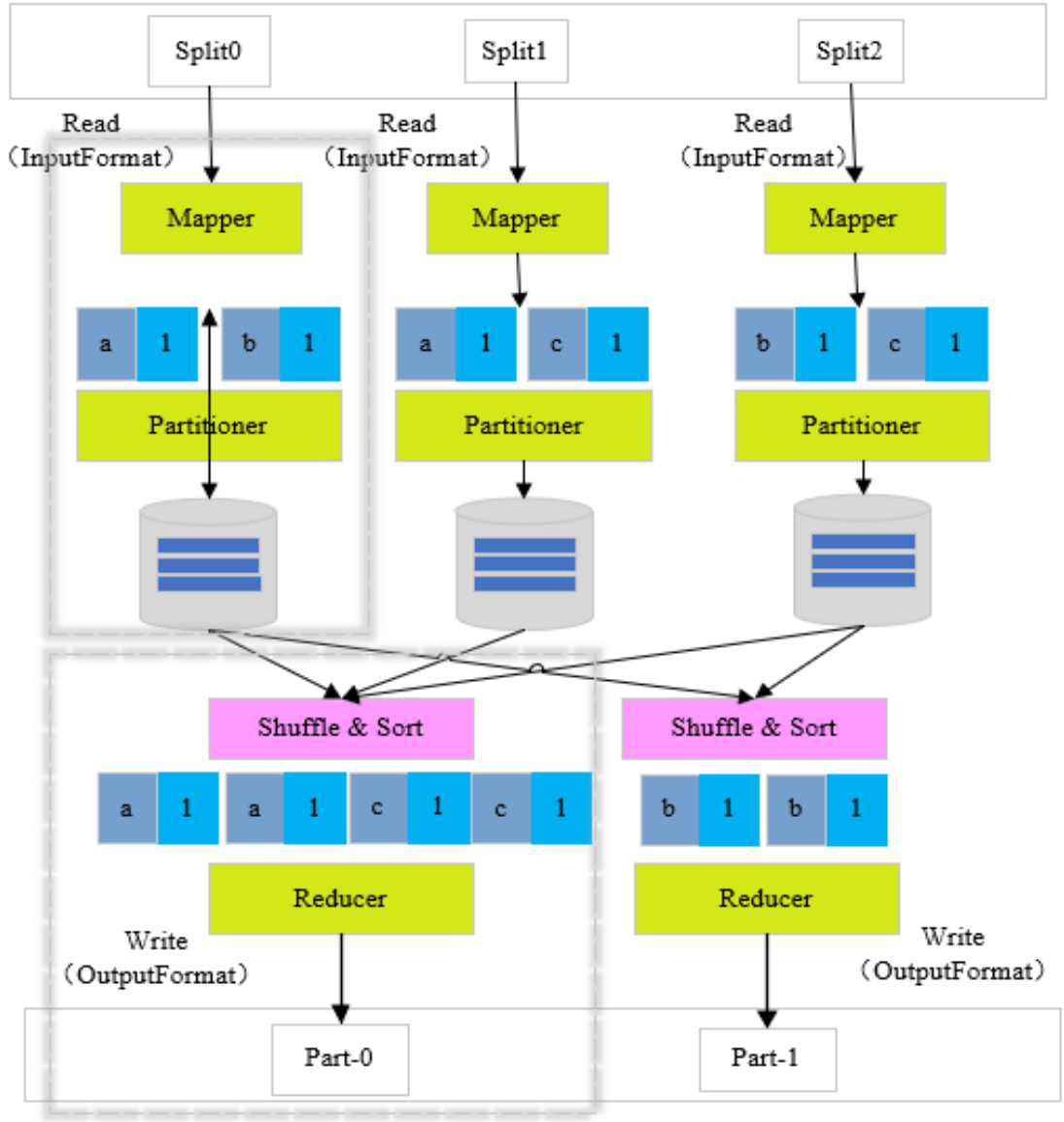
大纲

MapReduce编程模型

MapReduce编程模型

- MapReduce将作业的整个运行过程分为两个阶段Map阶段和Reduce阶段
- Map阶段由一定数量的Map Task组成
  - 输入数据格式解析：InputFormat
  - 输入数据处理：Mapper
  - 数据分组：Partitioner
- Reduce阶段由一定数量的Reduce Task组成
  - 数据远程拷贝
  - 数据按照key排序
  - 数据处理：Reducer
  - 数据输出格式：OutputFormat

MapReduce内部逻辑



MapReduce编程模型-InputFormat

- 文件分片（InputSplit）方法
  - 处理跨行问题
- 将分片数据解析成key/value对
  - 默认实现是TextInputFormat
- TextInputFormat
  - Key是行在文件中的偏移量，Value是行内容
  - 如果一行被截断，则读取下一个block的前几个字符

MapReduce编程模型-Split与Block

- Block
  - HDFS中最小数据处理单位
  - 默认128MB
- Split
  - MapReduce中最小的计算单元
  - 默认与Block一一对应
- Block与Split
  - Split与Block的对应关系是任意的，可由用户控制

MapReduce编程模型-Partitioner

- Partitioner决定了Map Task输出的每条数据交给哪个Reduce Task处理
- 默认实现：HashPartitioner
  - Hash(key) mod R
  - R是Reduce Task数目
  - 分区计算结果产生的分区号等于Reduce Task号
- 允许自定义分区

大纲

编程接口

Java编程接口

- Java编程接口组成
  - 旧API:所在java包:org.apache.hadoop.mapred
  - 新API:所在java包:org.apache.hadoop.mapreduce
- 新API具有更好的扩展性
- 两种编程接口只是暴露给用户的形式不同，内部执行引擎是一样的

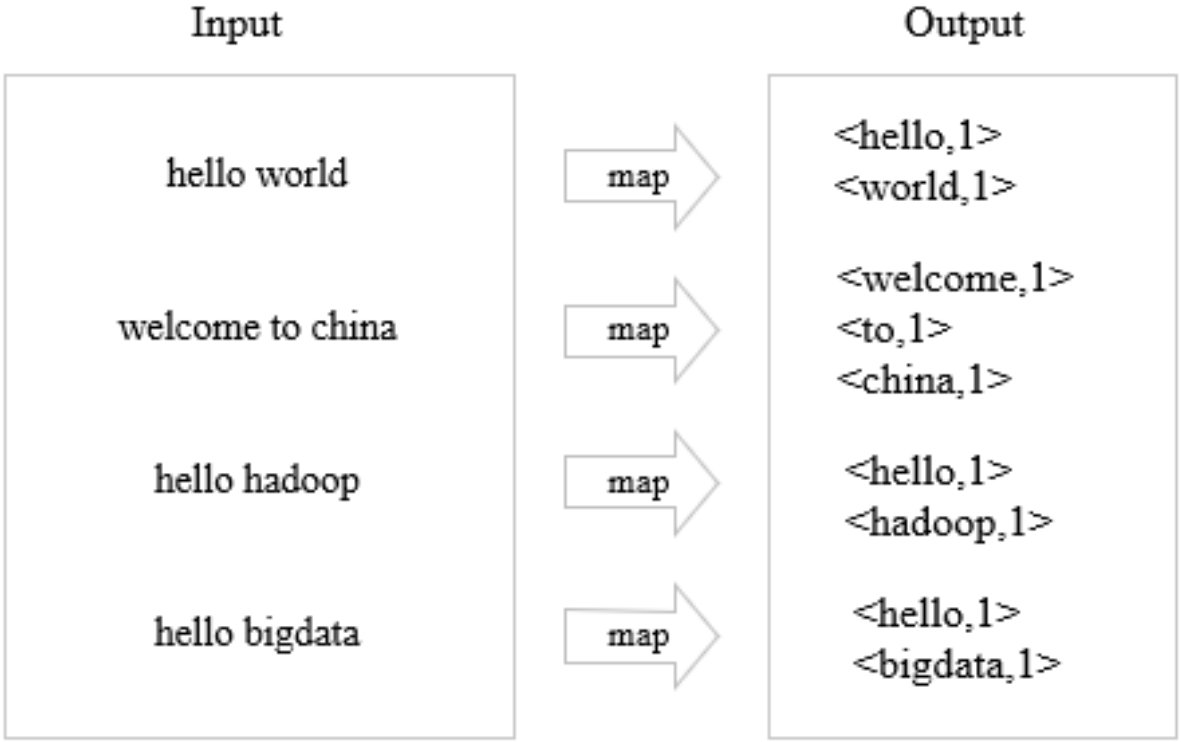
WordCount需求

需求

- 有TB级数据量的文件，文件中存储的各种各样的单词，统计在这些文件中每个单词的出现次数

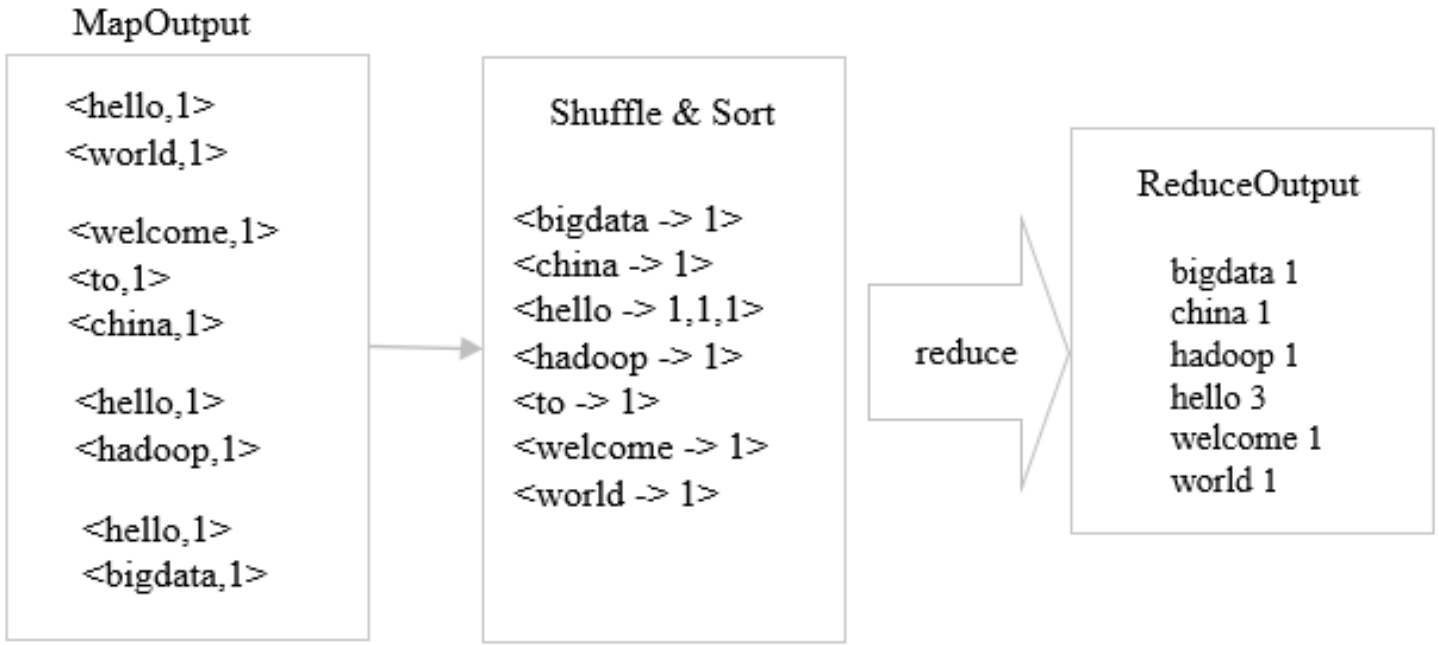
WordCount需求分析-Map阶段

1. TextInputFormat将文本数据解析成key-value对，Key是偏移量，value是行内容
2. 针对每行数据应用map方法，按照分隔符拆分
3. Map阶段输出单词作为key，1作为value



WordCount需求分析-Reduce阶段

1. 远程从Map阶段的输出结果中把数据拷贝到Reducer中
2. 对拷贝过来的数据整理排序，将相同key的数据整合成一个key对应一组value形式的数据集
3. 针对每个key对应的数据集调用reduce方法，对数据集中的value求和



WordCount设计与实现

- 继承org.apache.hadoop.mapreduce.Mapper类，编写自己的Mapper实现类，重写map方法，实现具体处理逻辑
- 继承org.apache.hadoop.mapreduce.Reducer类，编写自己的Reducer实现类，重写reduce方法，实现具体的处理逻辑
- 编写Driver，设置Job属性
- 打包提交运行jar包

  hadoop jar jar包路径 主类路径 输入路径 输出路径

  注意：输出路径在提交之前要保证不存在，MapReduce运行完会自动创建，如果存在输出路径在提交的时候会报文件夹存在的错误提示

MapReduce编程规则

1. 用户编写的程序分成三个部分：Mapper，Reducer，Driver(提交运行mr程序的客户端)
2. Mapper的输入数据是KV对的形式
3. Mapper的输出数据是KV对的形式
4. Mapper中的业务逻辑写在map()方法中
5. map()方法对每一个<K,V>调用一次
6. Reducer的输入数据类型对应Mapper的输出数据类型，也是KV对
7. Reducer的业务逻辑写在reduce()方法中
8. Reducetask进程对每一组相同k的<k,v>组调用一次reduce()方法
9. 用户自定义的Mapper和Reducer都要继承各自的父类
10. 整个程序需要一个Drvier来进行提交，提交的是一个描述了各种必要信息的job对象

疑问

- 小象问答官网
  - http://wenda.chinahadoop.cn

联系我们

小象学院：互联网新技术在线教育领航者

- 微信公众号：小象学院
- 新浪微博：小象AI学院