Hadoop MapReduce基本架构

摘要

- Hadoop平台的基本组成与生态系统
- 分布式文件系统HDFS
- Hadoop MapReduce的基本工作原理

官网：https://hadoop.apache.org
官方文档 https://hadoop.apache.org/docs/stable/index.html

摘要

- Hadoop平台的基本组成与生态系统
- 分布式文件系统HDFS
- Hadoop MapReduce的基本工作原理

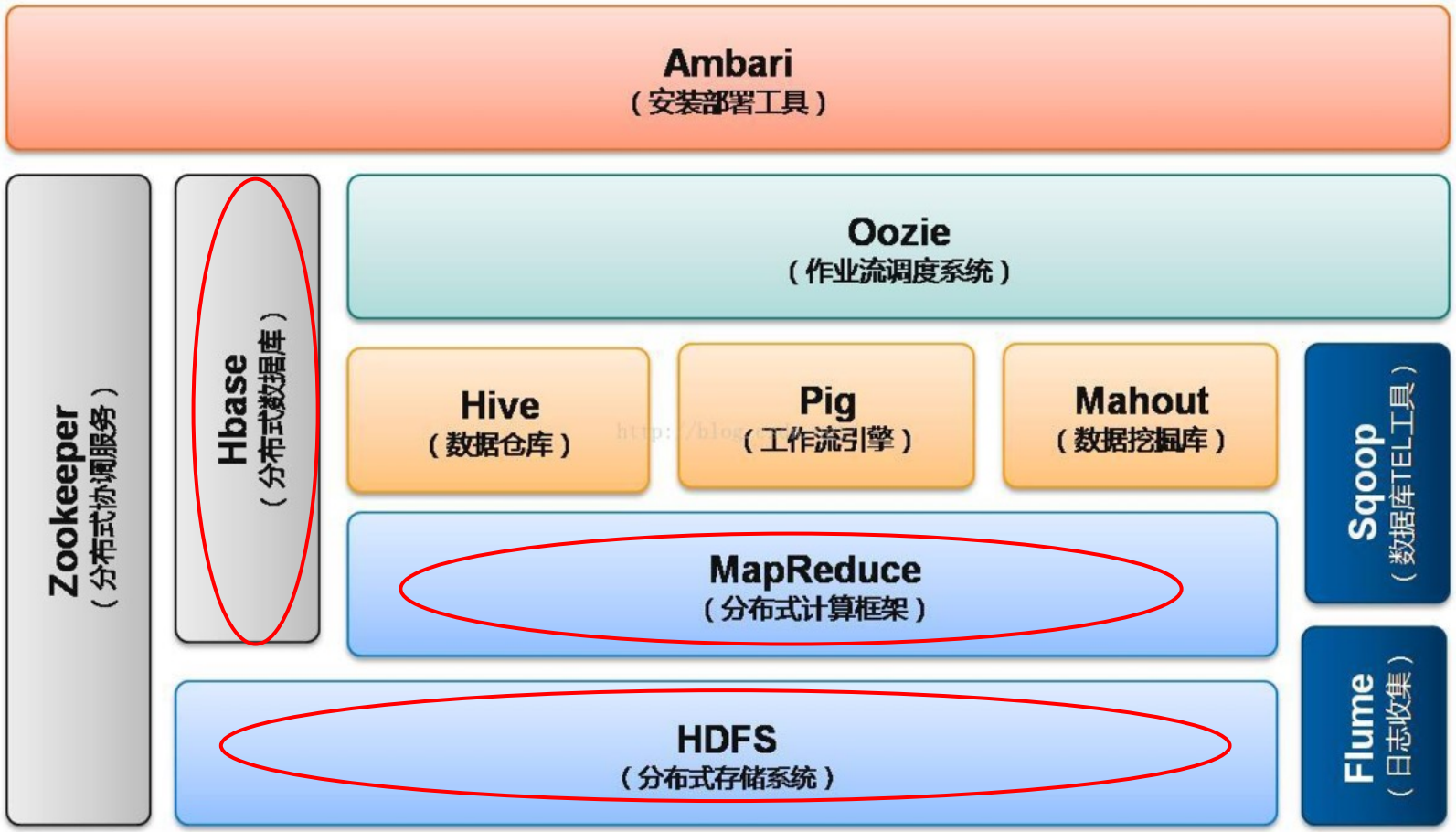
Hadoop简介

- 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。

Hadoop特性

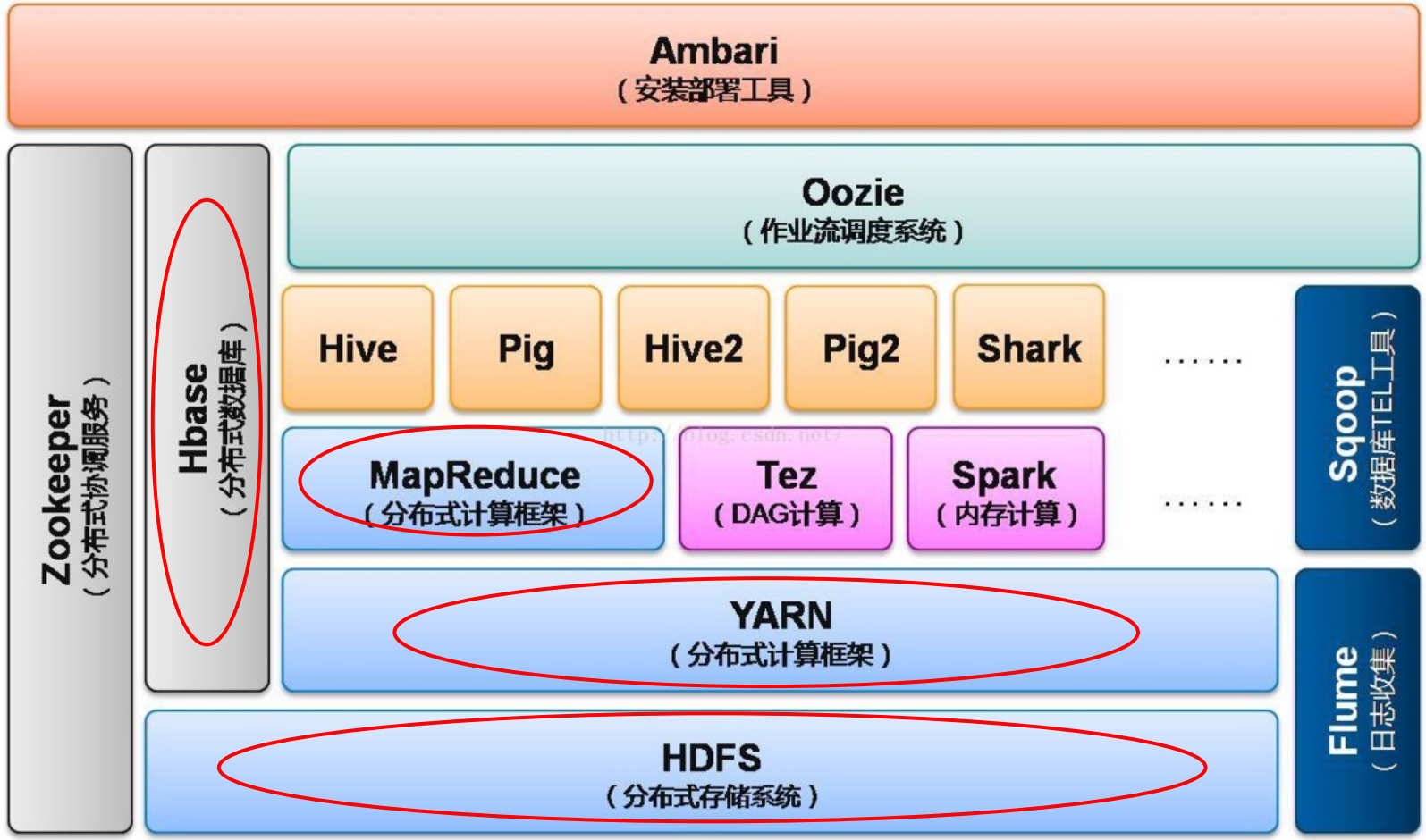
- Hadoop是一个能够对大量数据进行分布式处理的软件框架，并且是以一种可靠、高效、可伸缩的方式进行处理的，它具有以下几个方面的特性：
  - 高可靠性
  - 高效性
  - 高可扩展性
  - 高容错性
  - 成本低
  - 运行在Linux平台上
  - 支持多种编程语言

Hadoop基本组成和生态系统



Hadoop 1.x架构

Hadoop基本组成和生态系统

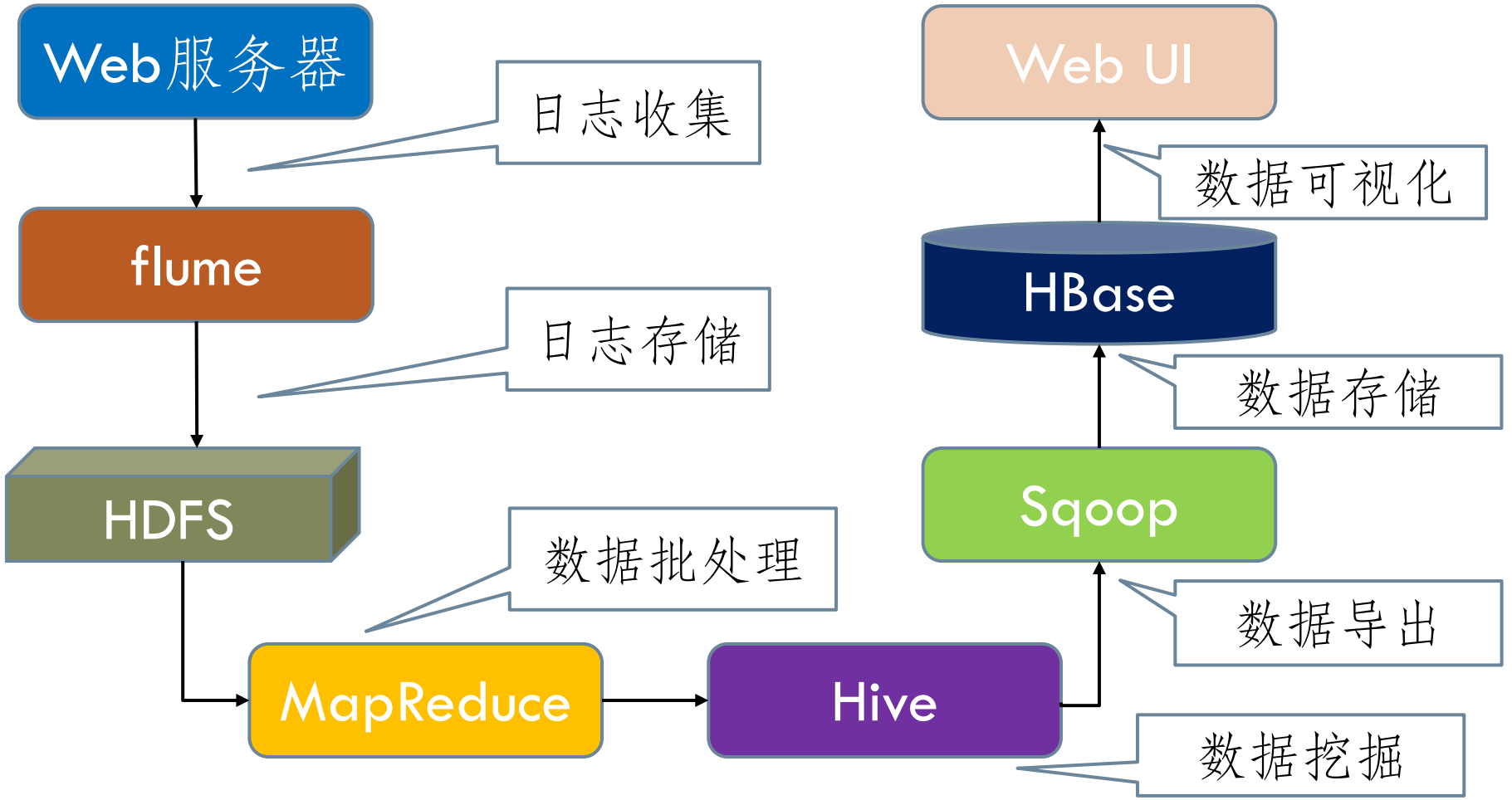


Hadoop 2.x架构

Hadoop基本组成和生态系统

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

常见大数据应用系统架构



摘要

- Hadoop平台的基本组成与生态系统
- 分布式文件系统HDFS
- Hadoop MapReduce的基本工作原理

Google GFS的基本设计原则

- Google GFS是一个基于分布式集群的大型分布式文件系统，为MapReduce计算框架提供数据存储和数据可靠性支撑；
- GFS是一个构建在分布节点本地文件系统之上的一个逻辑上文件系统，它将数据存储在物理上分布的每个节点上，但通过GFS将整个数据形成一个逻辑上整体的文件。

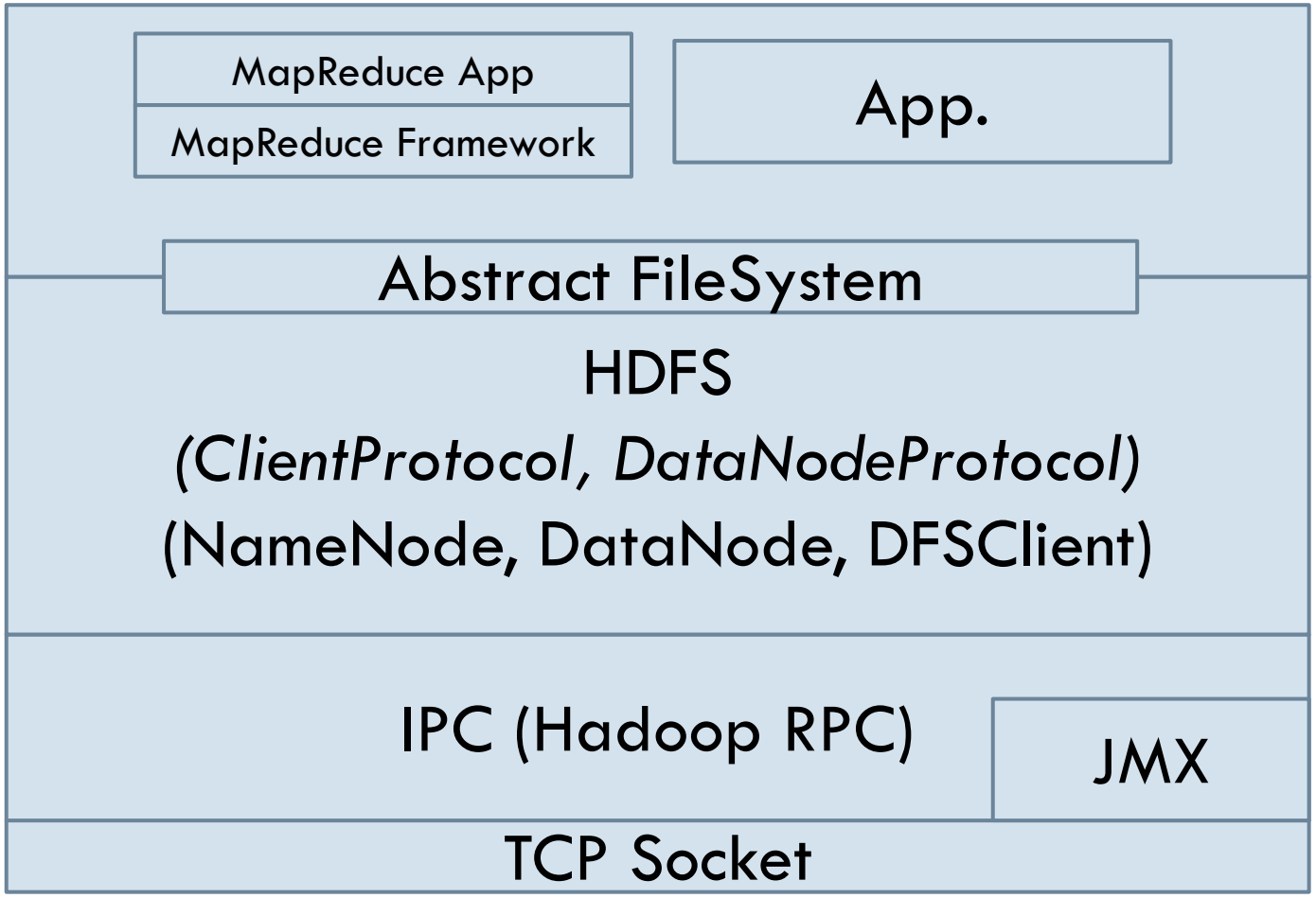
HDFS的基本特征

- 模仿Google GFS设计实现
- 存储极大数目的信息（terabytes or petabytes），将数据保存到大量的节点当中；支持很大的单个文件。
- 提供数据的高可靠性和容错能力，单个或者多个节点不工作，对系统不会造成任何影响，数据仍然可用。通过一定数量的数据复制保证数据存储的可靠性和出错恢复能力。
- 提供对数据的快速访问；并提供良好的可扩展性，通过简单加入更多服务器快速扩充系统容量，服务更多的客户端。
- 与GFS类似，HDFS是MapReduce的底层数据存储支撑，并使得数据尽可能根据其本地局部性进行访问与计算。

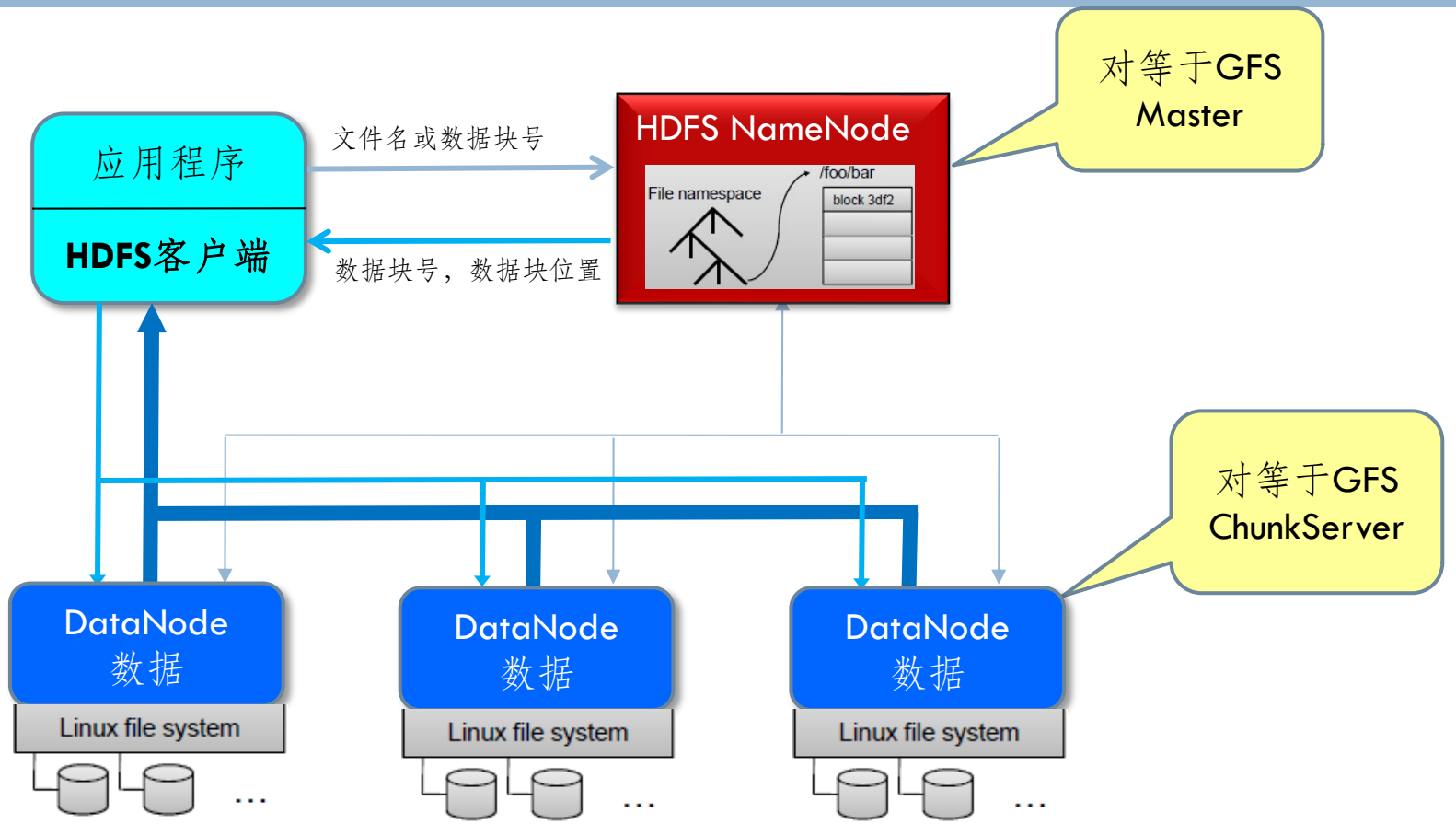
HDFS的基本特征

- HDFS对顺序读进行了优化，支持大量数据的快速顺序读出，代价是对于随机的访问负载较高。
- 数据支持一次写入，多次读取；不支持已写入数据的更新操作。
- 数据不进行本地缓存（文件很大，且顺序读没有局部性）
- 基于块的文件存储，默认的块的大小是64MB（1.x及之前）或者128MB（2.x及后续），可通过dfs.blocksize参数自定义块大小
  - 减少元数据的开销，提高吞吐量，适配分布式计算
  - 优化数据本地计算效率，有利于顺序读写（在磁盘上数据顺序存放）
- 多副本数据块形式存储，按照块的方式随机选择存储节点，默认副本数目是3

HDFS基本构架

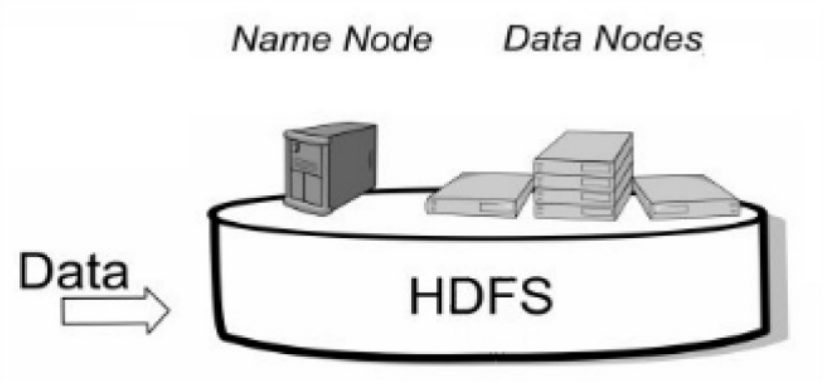

MapReduce App
MapReduce Framework
App.
Abstract FileSystem
HDFS
(ClientProtocol, DataNodeProtocol)
(NameNode, DataNode, DFSClient)
IPC (Hadoop RPC)
JMX
TCP Socket

HDFS基本构架


应用程序
HDFS客户端
文件名或数据块号
数据块号，数据块位置
HDFS NameNode
File namespace
/foo/bar
block 3df2
对等于GFS Master
DataNode 数据
Linux file system
DataNode 数据
Linux file system
DataNode 数据
Linux file system
对等于GFS ChunkServer

HDFS基本构架

HDFS主要组件的功能



metadata

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

Blk B:
DN7, DN1, DN2

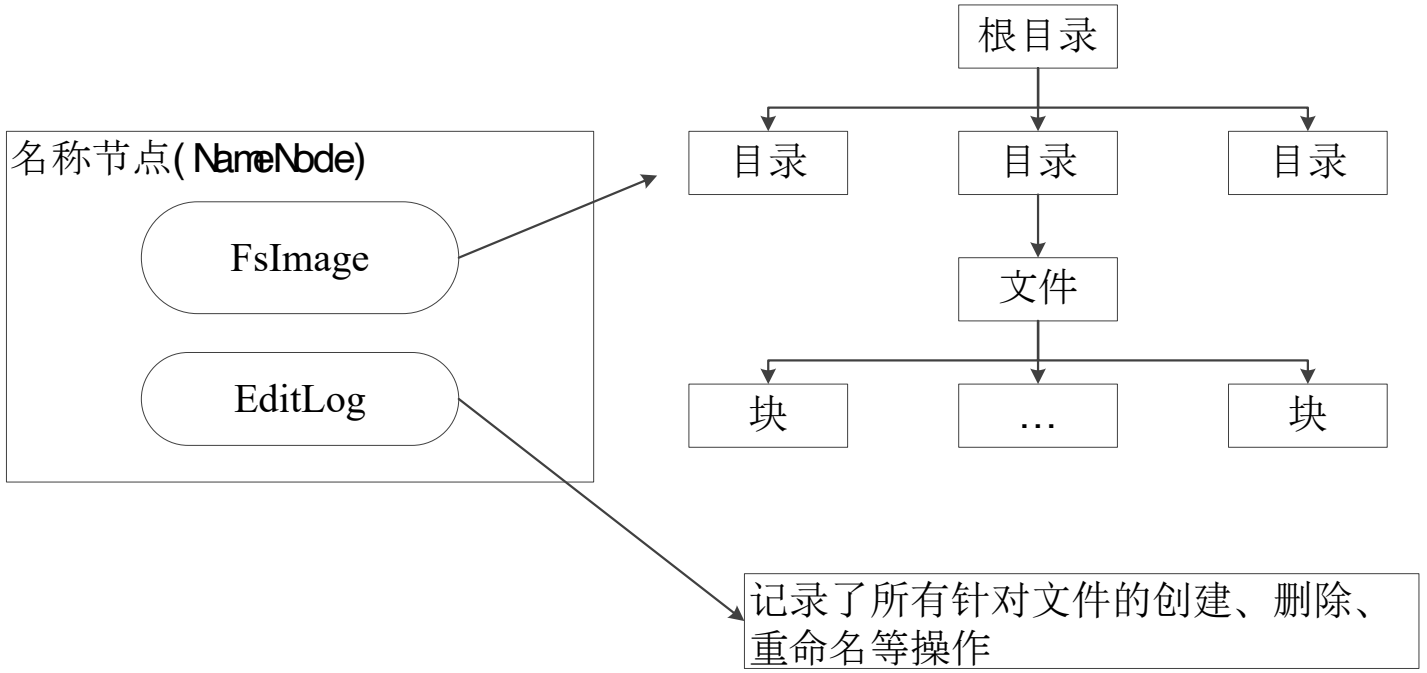
Blk C:
DN5, DN8,DN9

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

NameNode

- 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

NameNode


名称节点(NameNode)
FsImage
EditLog
根目录
目录
目录
目录
文件
块
…
块
记录了所有针对文件的创建、删除、
重命名等操作

NameNode

- FsImage文件
  - FsImage文件包含文件系统中所有目录和文件inode的序列化形式。每个inode是一个文件或目录的元数据的内部表示，并包含此类信息：文件的复制等级、修改和访问时间、访问权限、块大小以及组成文件的块。对于目录，则存储修改时间、权限和配额元数据。
  - FsImage文件没有记录文件包含哪些块以及每个块存储在哪个数据节点。而是由名称节点把这些映射信息保留在内存中，当数据节点加入HDFS集群时，数据节点会把自己所包含的块列表告知给名称节点，此后会定期执行这种告知操作，以确保名称节点的块映射是最新的。

NameNode

- 启动过程
  - 在名称节点启动的时候，它会将FsImage文件中的内容加载到内存中，之后再执行EditLog文件中的各项操作，使得内存中的元数据和实际的同步，存在内存中的元数据支持客户端的读操作。
  - 一旦在内存中成功建立文件系统元数据的映射，则创建一个新的FsImage文件和一个空的EditLog文件。
  - 名称节点起来之后，HDFS中的更新操作会重新写到EditLog文件中，因为FsImage文件一般都很大（GB级别的很常见），如果所有的更新操作都往FsImage文件中添加，这样会导致系统运行的十分缓慢，但是，如果往EditLog文件里面写就不会这样，因为EditLog 要小很多。每次执行写操作之后，且在向客户端发送成功代码之前，edits文件都需要同步更新。

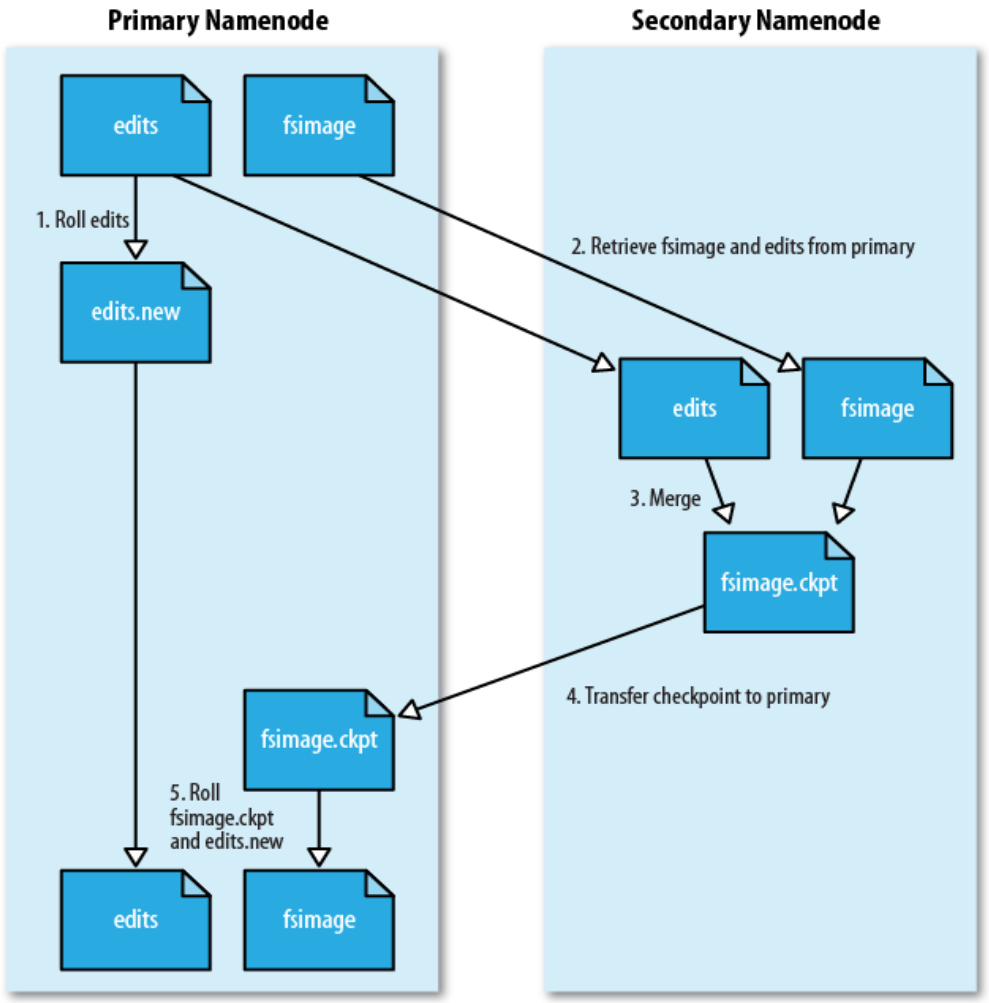
NameNode

- 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。

SecondaryNameNode

- **SecondaryNameNode**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有两个作用：1. 镜像备份；2. 日志与镜像的定期合并。两个过程同时进行，称为checkpoint。

SecondaryNameNode



- 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）

SecondaryNameNode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

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。

HDFS命名空间管理

- HDFS的命名空间包含目录、文件和块。
- 在HDFS1.0体系结构中，在整个HDFS集群中只有一个命名空间，并且只有唯一一个名称节点，该节点负责对这个命名空间进行管理。
- HDFS使用的是传统的分级文件体系，因此，用户可以像使用普通文件系统一样，创建、删除目录和文件，在目录间转移文件，重命名文件等。

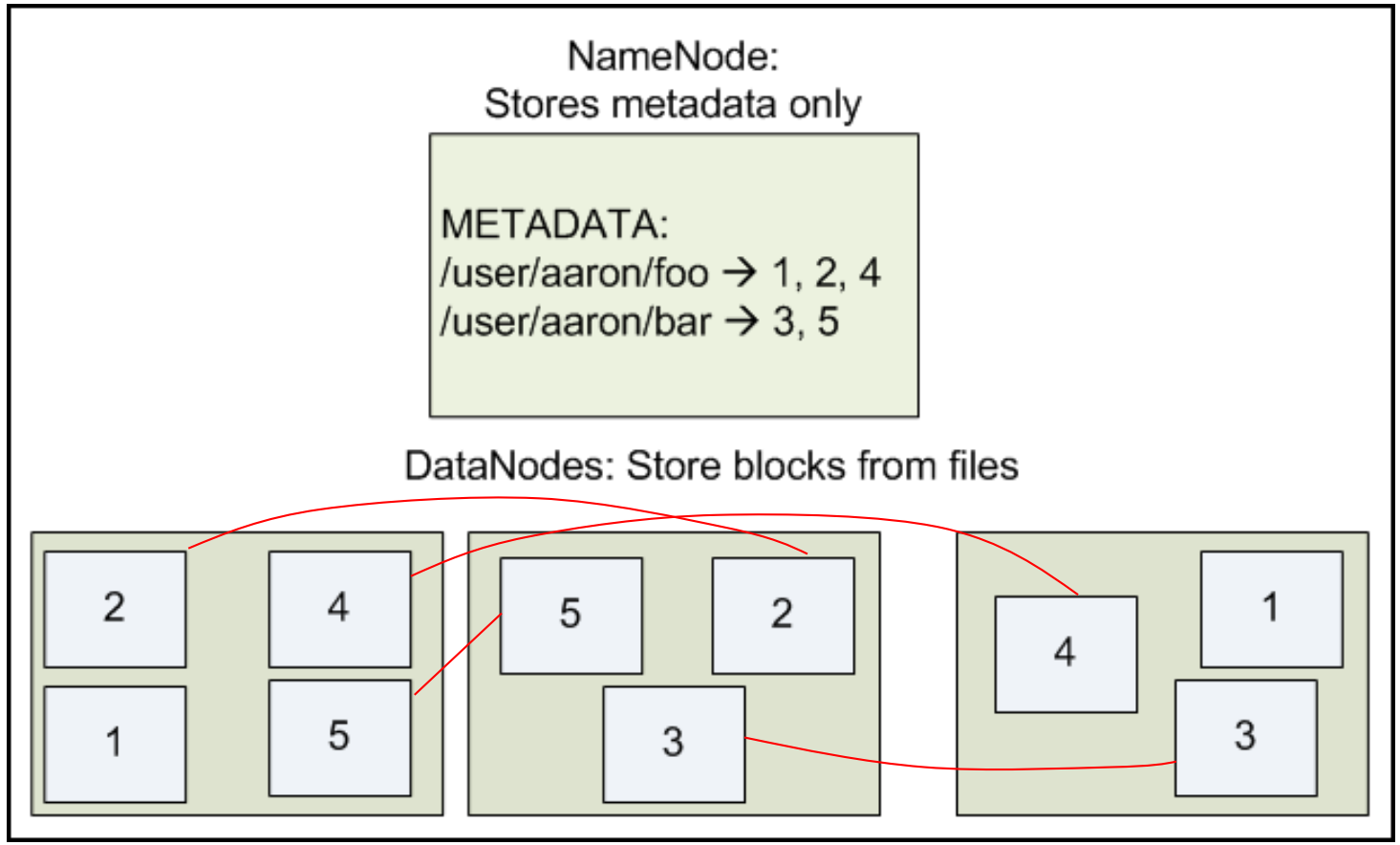
HDFS通信协议

- HDFS是一个部署在集群上的分布式文件系统，因此，很多数据需要通过网络进行传输。
- 所有的HDFS通信协议都是构建在TCP/IP协议基础之上的
- 客户端通过一个可配置的端口向NameNode主动发起TCP连接，并使用客户端协议与NameNode进行交互。
- NameNode和DataNode之间则使用数据节点协议进行交互。
- 客户端与DataNode的交互是通过RPC（Remote Procedure Call）来实现的。在设计上，NameNode不会主动发起RPC，而是响应来自客户端和DataNode的RPC请求。

HDFS客户端

- 客户端是用户操作HDFS最常用的方式，HDFS在部署时都提供了客户端。
- HDFS客户端是一个库，暴露了HDFS文件系统接口，这些接口隐藏了HDFS实现中的大部分复杂性。
- 严格来说，客户端并不算是HDFS的一部分。
- 客户端可以支持打开、读取、写入等常见的操作，并且提供了类似Shell的命令行方式来访问HDFS中的数据。
- 此外，HDFS也提供了Java API，作为应用程序访问文件系统的客户端编程接口。

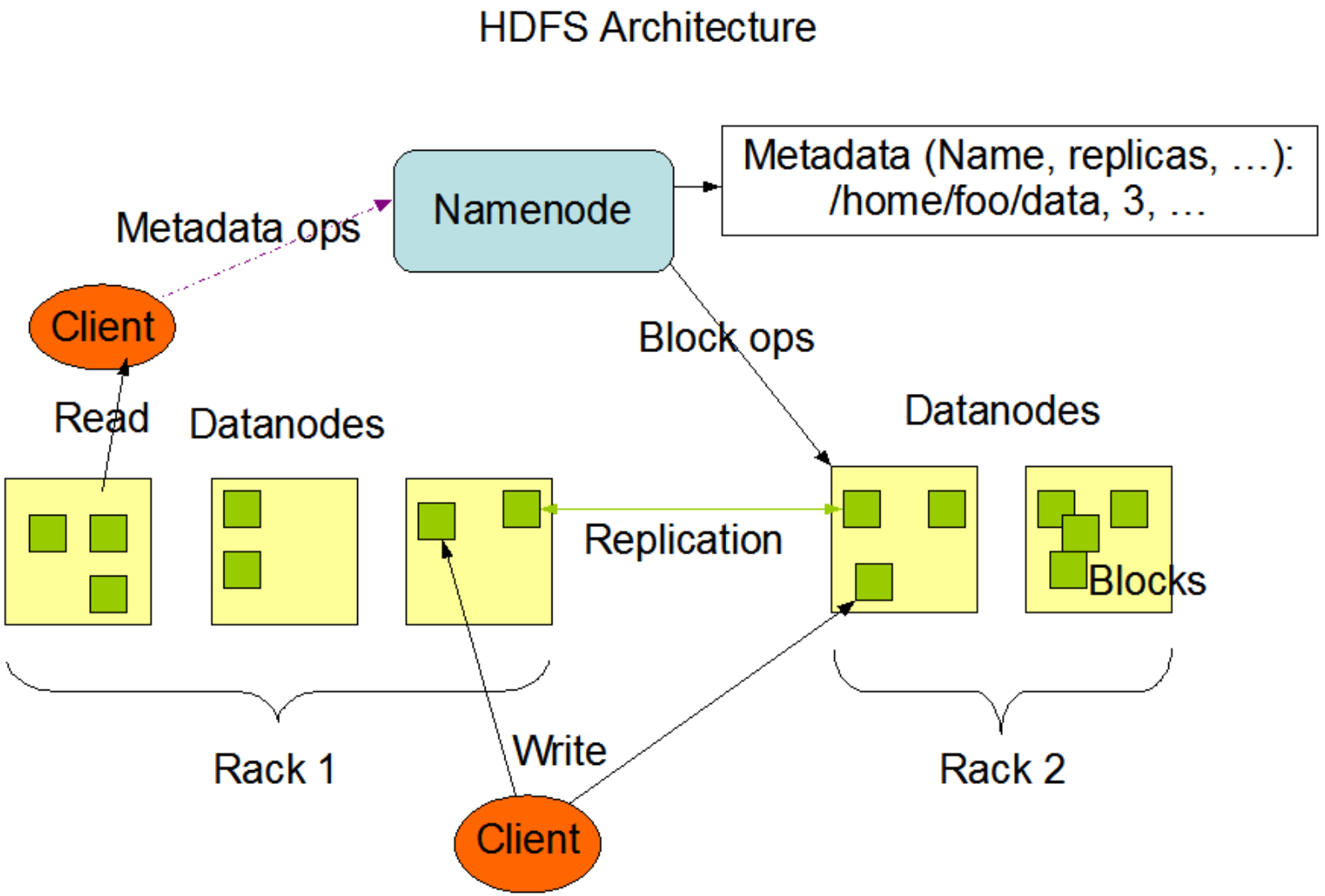
HDFS数据分布设计



多副本数据块形式存储，按照块的方式随机选择存储节点

默认副本数目是3

HDFS数据分布设计


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

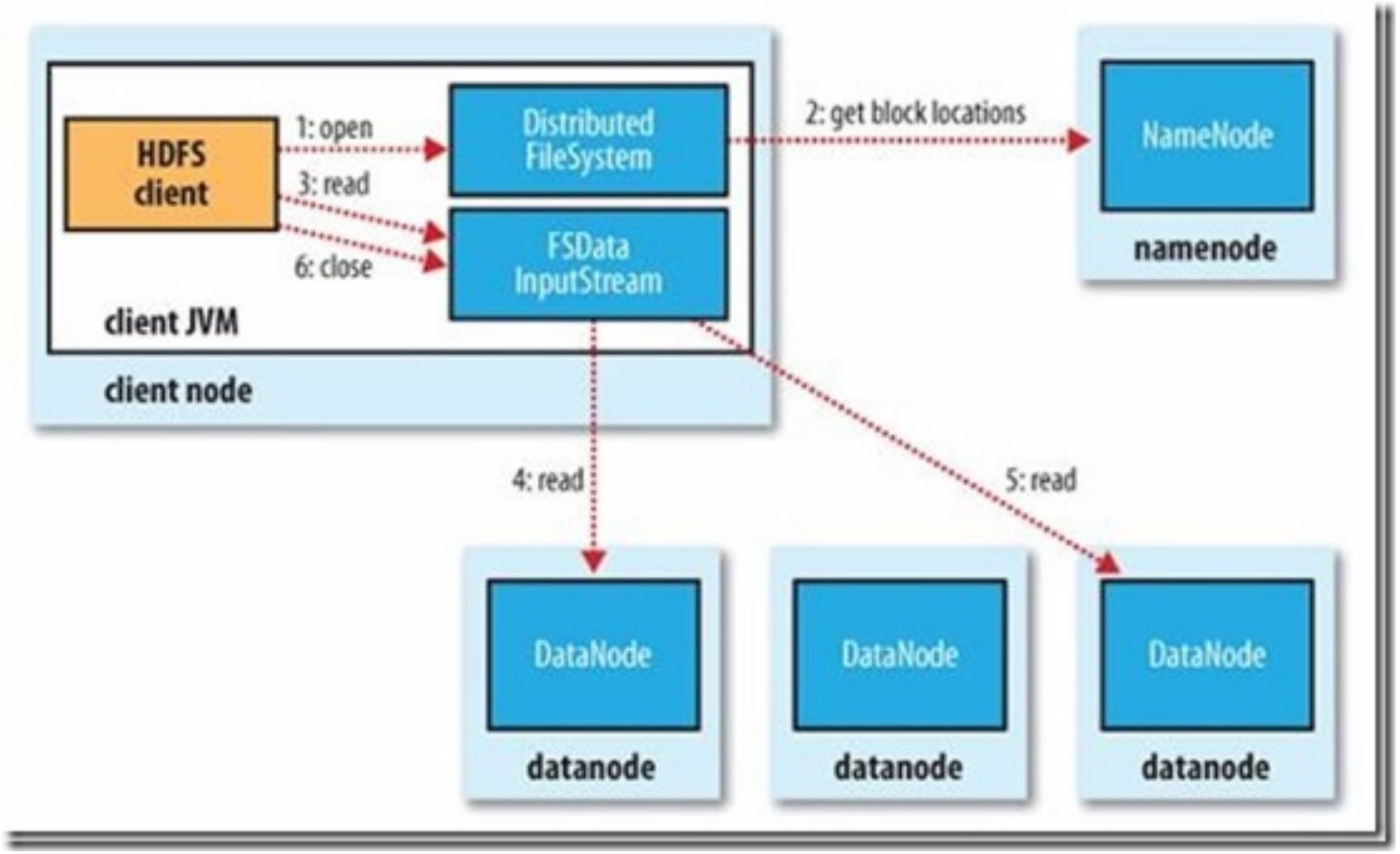
数据存取策略

- 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

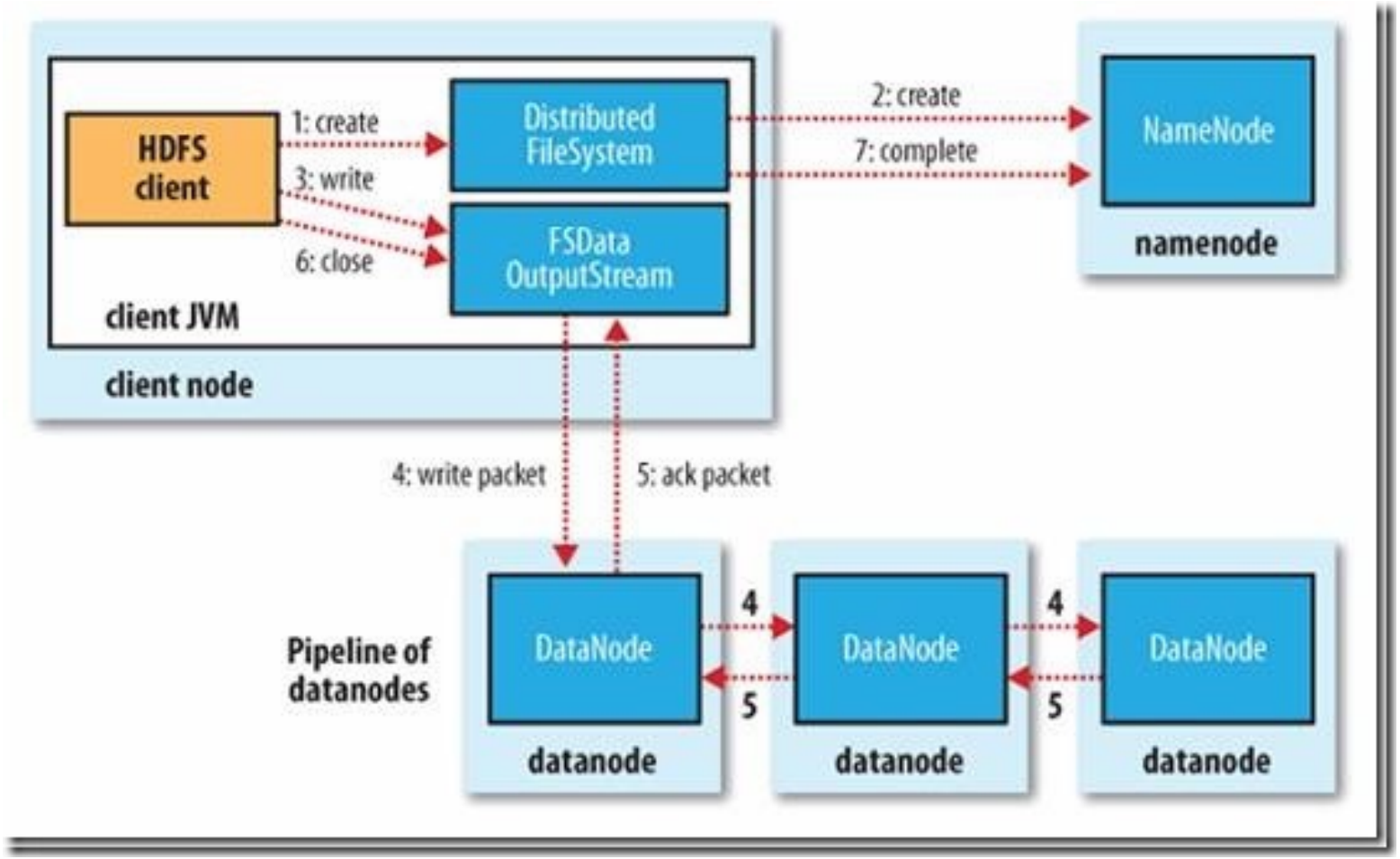
数据存取策略

- 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（Rack Awareness）。
  - 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。

HDFS读过程


HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode

HDFS写过程


HDFS client
1: create
3: write
6: close
Distributed FileSystem
FSData OutputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
DataNode
DataNode
4
5
4
5
datanode
datanode
datanode

HDFS可靠性与出错恢复

- DataNode节点的检测
  - 心跳：NameNode不断检测DataNode是否有效
  - 若失效，则寻找新的节点替代，将失效节点数据重新分布
- 集群负载均衡
- 数据一致性：校验和(checksum)
- 主节点元数据失效
  - Multiple FsImage and EditLog
  - Checkpoint

NameNode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。

DataNode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。

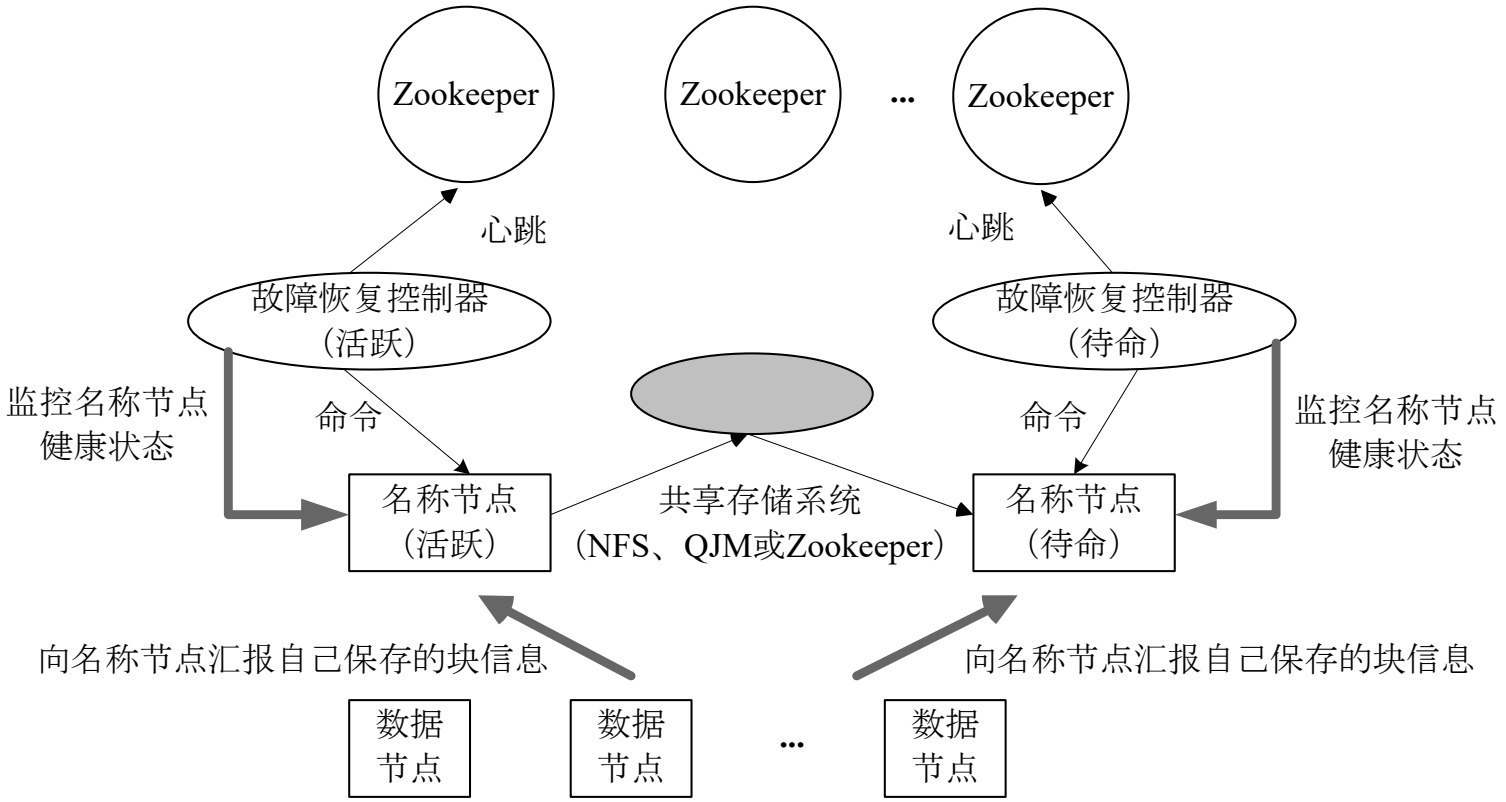
Hadoop v2.x vs. Hadoop v3.x

	Hadoop 2.x	Hadoop 3.x
JDK	Java 7 +	Java 8 +
容错	复制	Erasure编码
存储	200%开销	50%开销
YARN时间线服务	可伸缩的旧时间轴服务	改进时间线服务v2
兼容的文件系统	HDFS，FTP， Amazon S3， WASB	+ 微软Azure Data Lake
NameNode	2个	2个或更多个

HDFS HA

- HDFS HA（High Availability）是为了解决单点故障问题
- HA集群设置两个NameNode，“活跃（Active）”和“待命（Standby）”
- 两个NameNode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NameNode出现故障，就可以立即切换到待命NameNode
- Zookeeper确保一个NameNode在对外服务
- NameNode维护映射信息，DataNode同时向两个NameNode汇报信息

HDFS HA

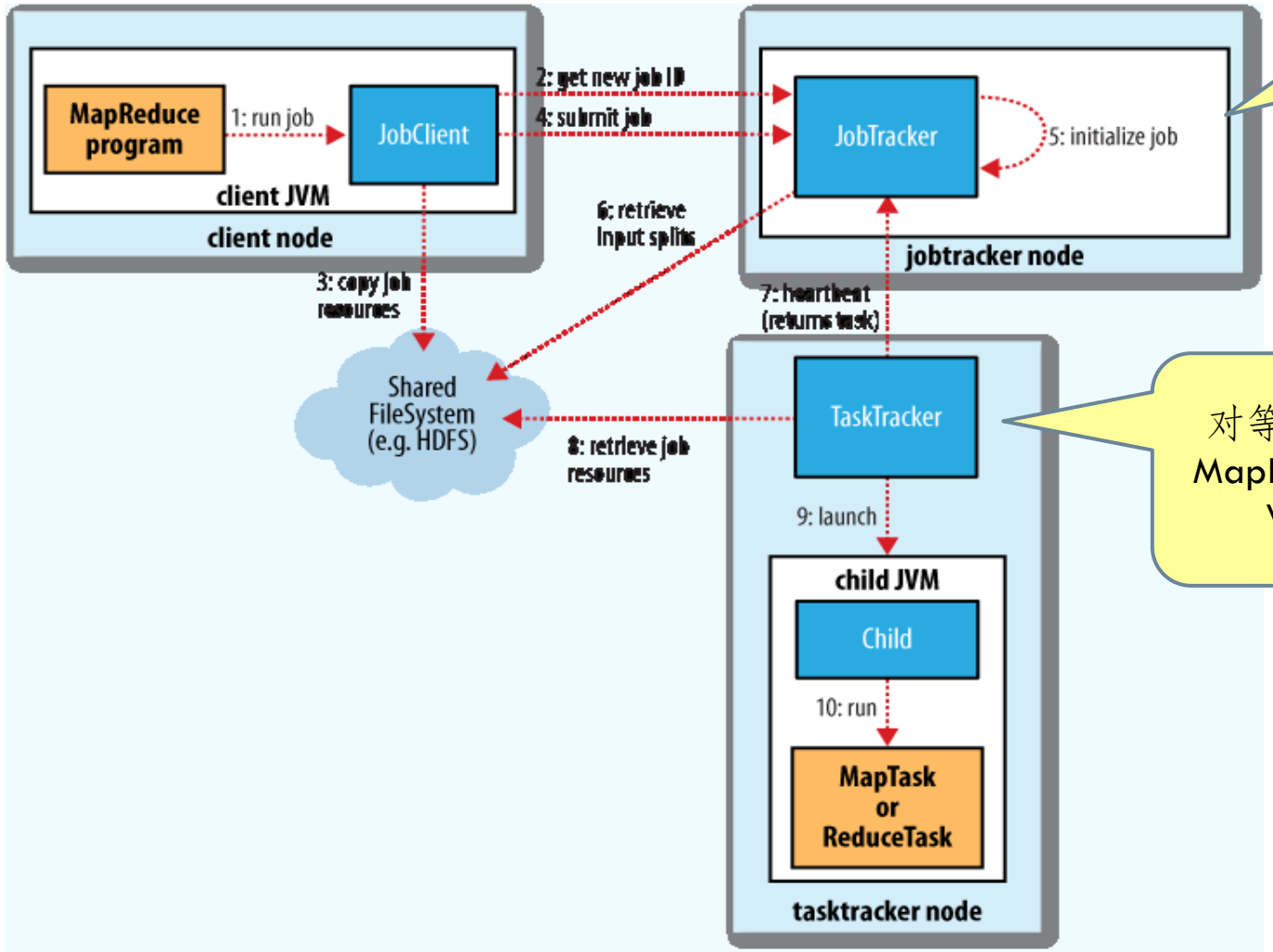


HDFS HA架构

摘要

- Hadoop平台的基本组成与生态系统
- 分布式文件系统HDFS
- Hadoop MapReduce的基本工作原理

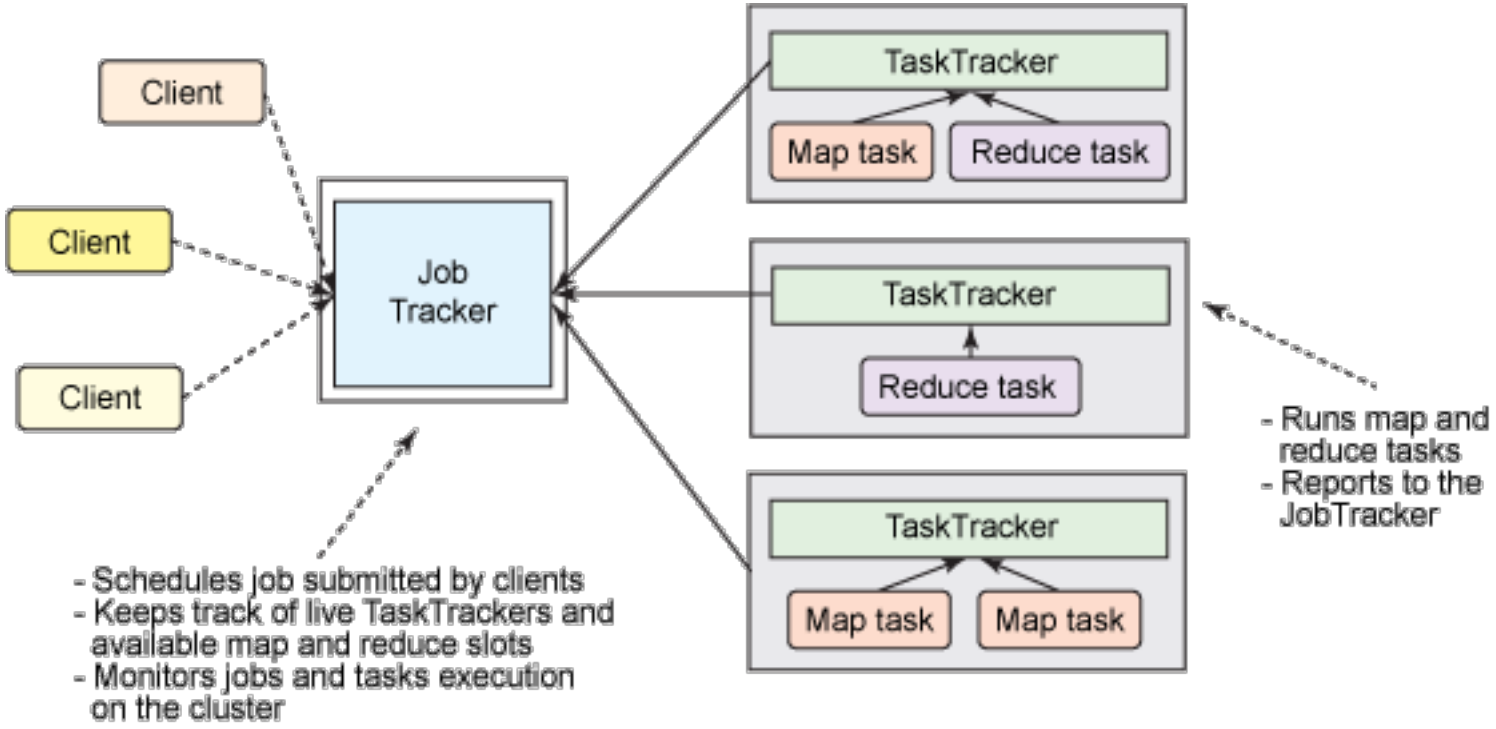
经典版的MapReduce架构（v1.0）



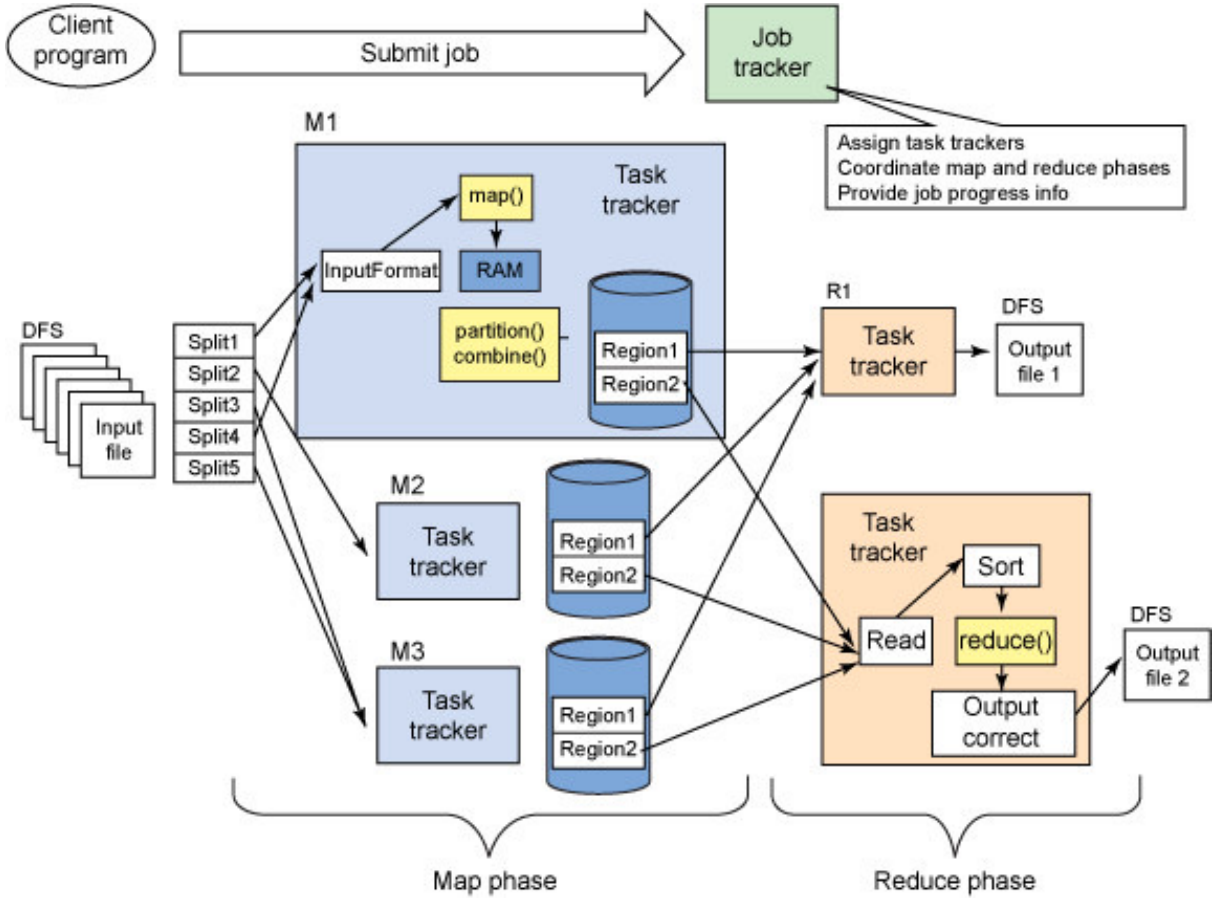
对等于Google MapReduce中的Master

对等于Google MapReduce中的Worker

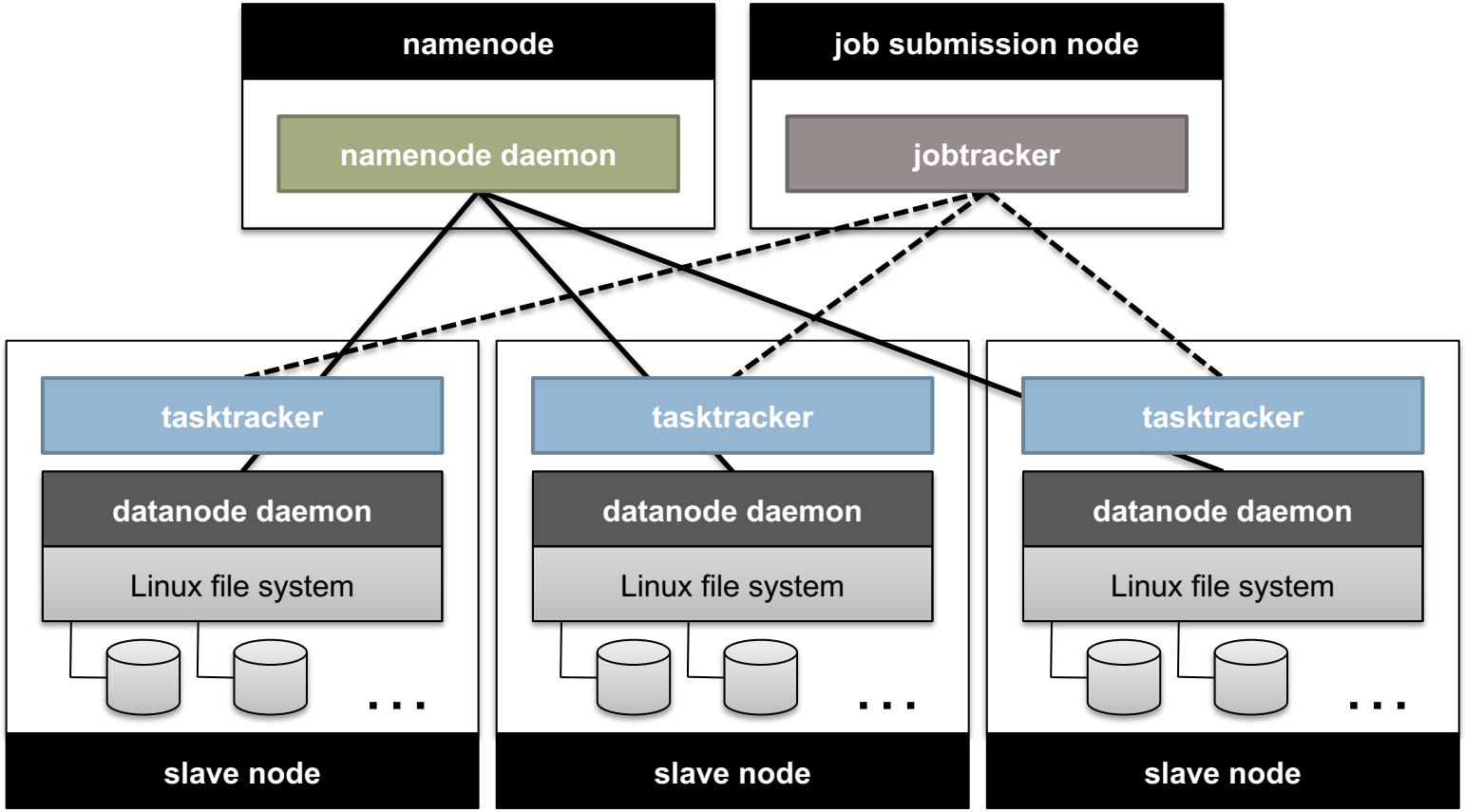
经典版的MapReduce架构（v1.0）



经典版的MapReduce架构（v1.0）



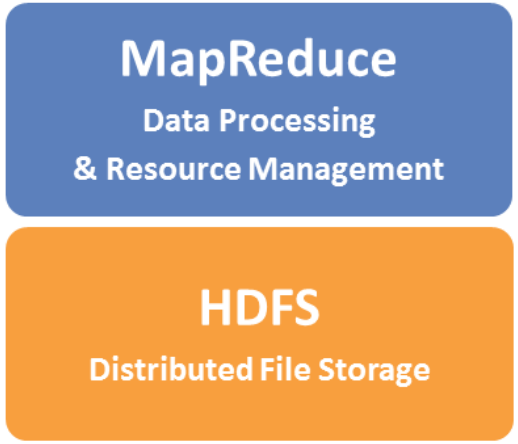
经典版的MapReduce架构（v1.0）

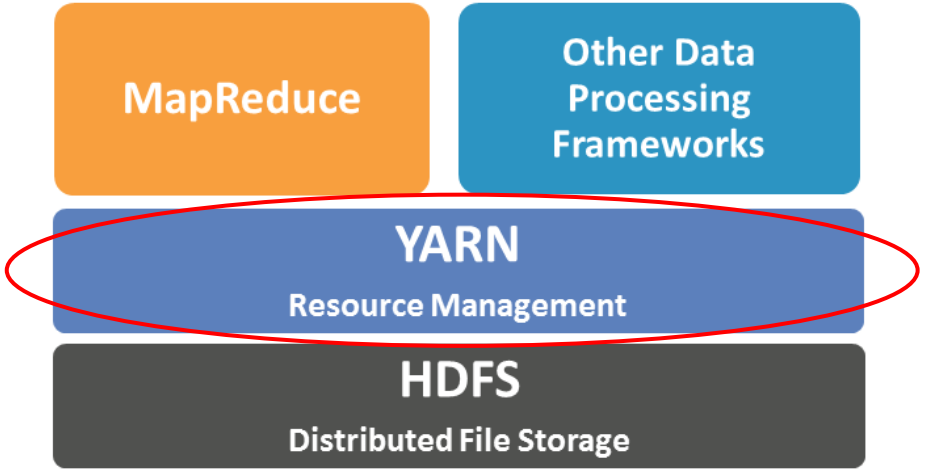


经典版的MapReduce架构（v1.0）

- 存在的问题
  - 单点故障：所有Job由JobTracker调度和分配
  - 可扩展性： JobTracker任务繁重，任务多时内存开销大
  - 容易出现内存溢出：TaskTracker端，资源的分配并不考虑CPU、内存的实际使用情况，而是根据任务个数分配资源
  - 资源分配不合理：资源被强制等量划分成多个“槽”(Slot)，slot又被进一步划分为Map slot和Reduce slot，彼此之间不能使用分配给对方的slot。

Hadoop v1.0 vs. Hadoop v2.0



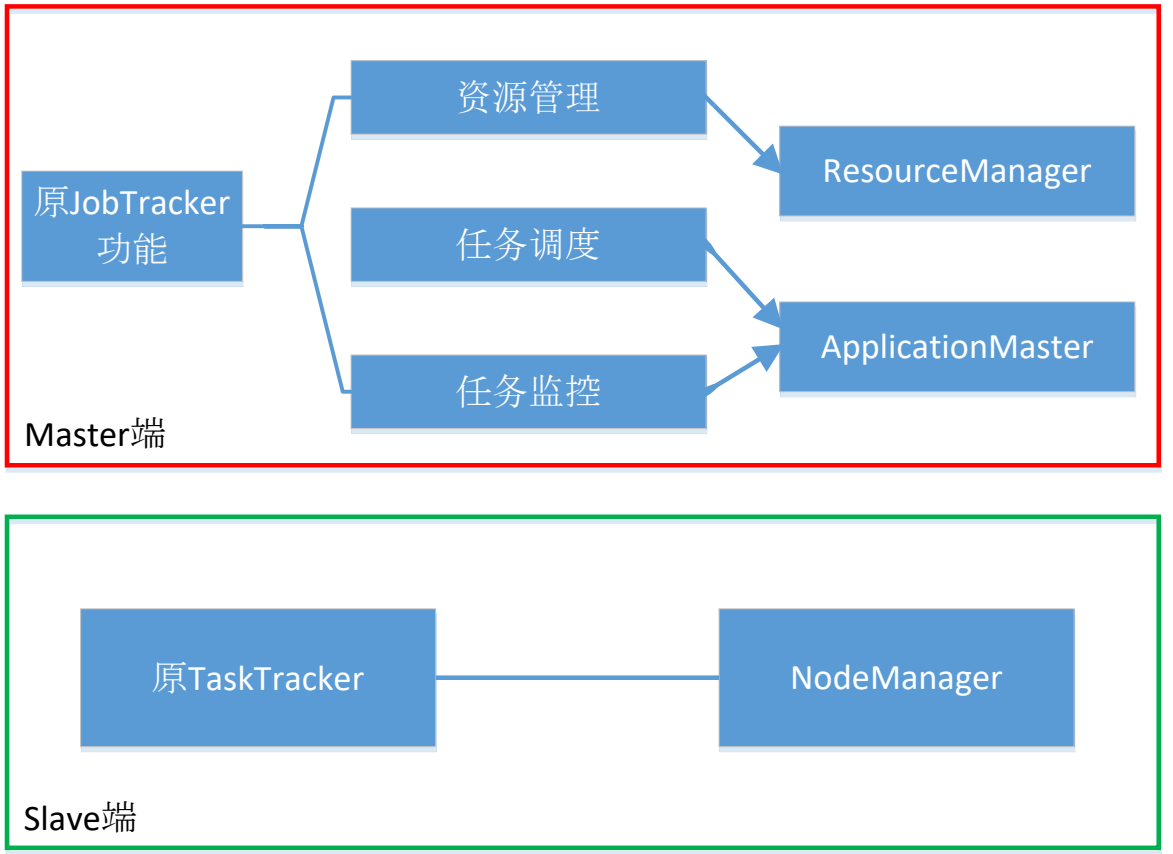


新一代的架构设计YARN（v2.0）

- Yet Another Resource Negotiator：另一种资源协调者。它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。一个框架未使用的资源可由另一个框架进行使用，充分的避免资源浪费。
- 在新的YARN中，ResourceManager (*RM*)全局管理所有应用程序计算资源的分配，每个应用的 ApplicationMaster (*AM*)负责相应的调度和协调。在上一版框架中，JobTracker一个很大的负担就是监控Job的tasks运行情况，现在，这个部分下放到了AM中。
- AM是一个可变更的部分，用户可以针对不同的编程模型编写自己的AM，让更多的编程模型运行在Hadoop集群中。

YARN设计思路

YARN架构思路：将原JobTacker三大功能拆分

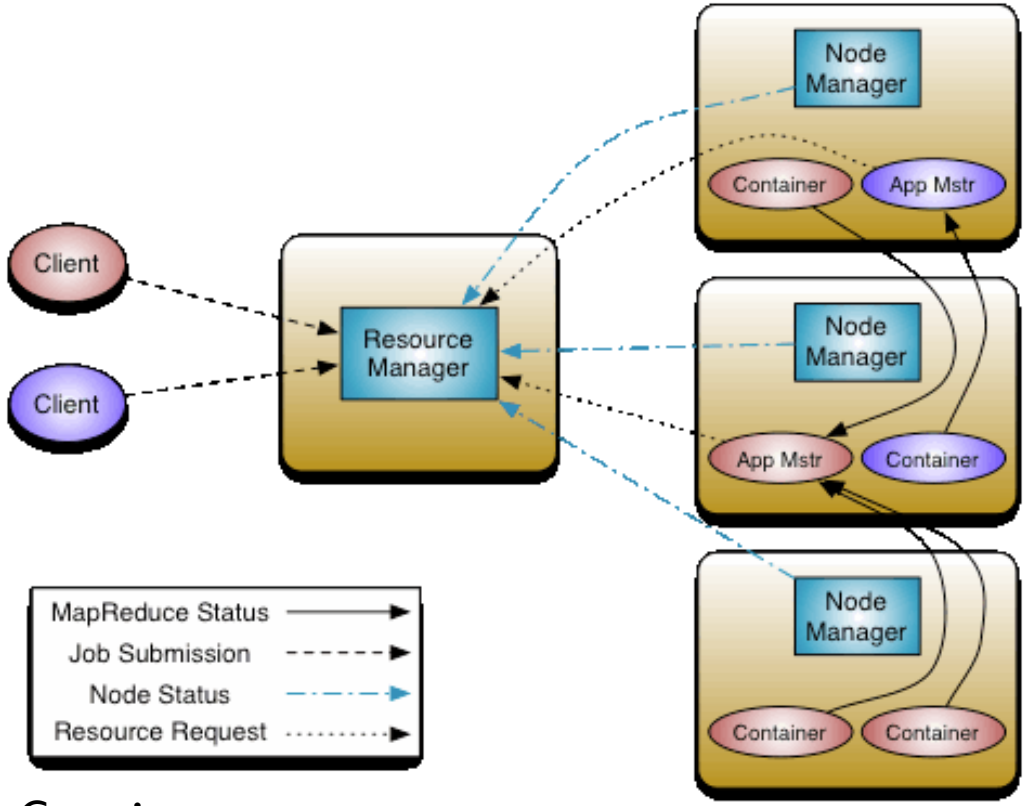


YARN Architecture （v2.0）

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

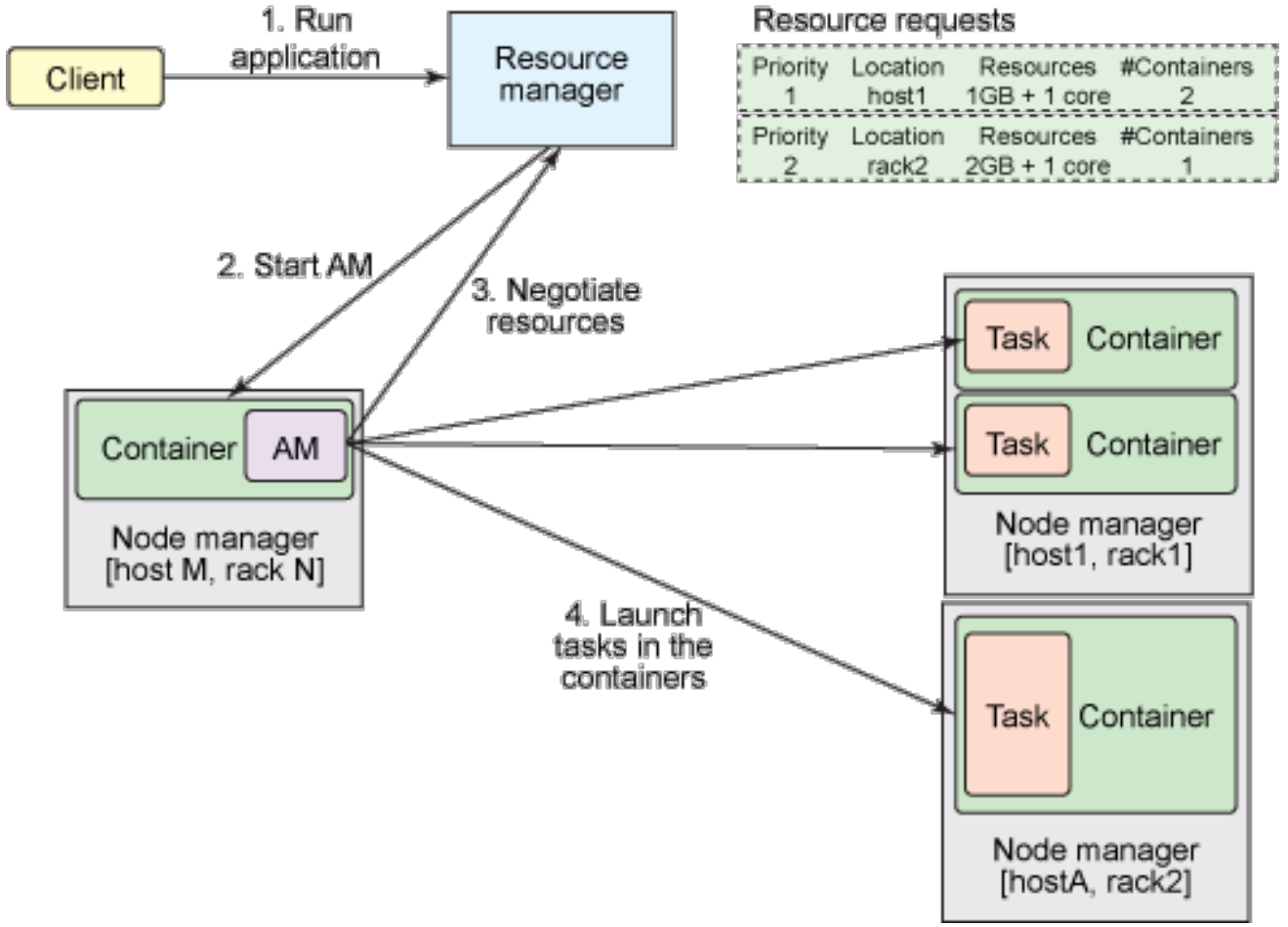
ApplicationMaster
- 为应用程序申请资源，并分配给内部任务
- 任务调度、监控与容错

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



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。

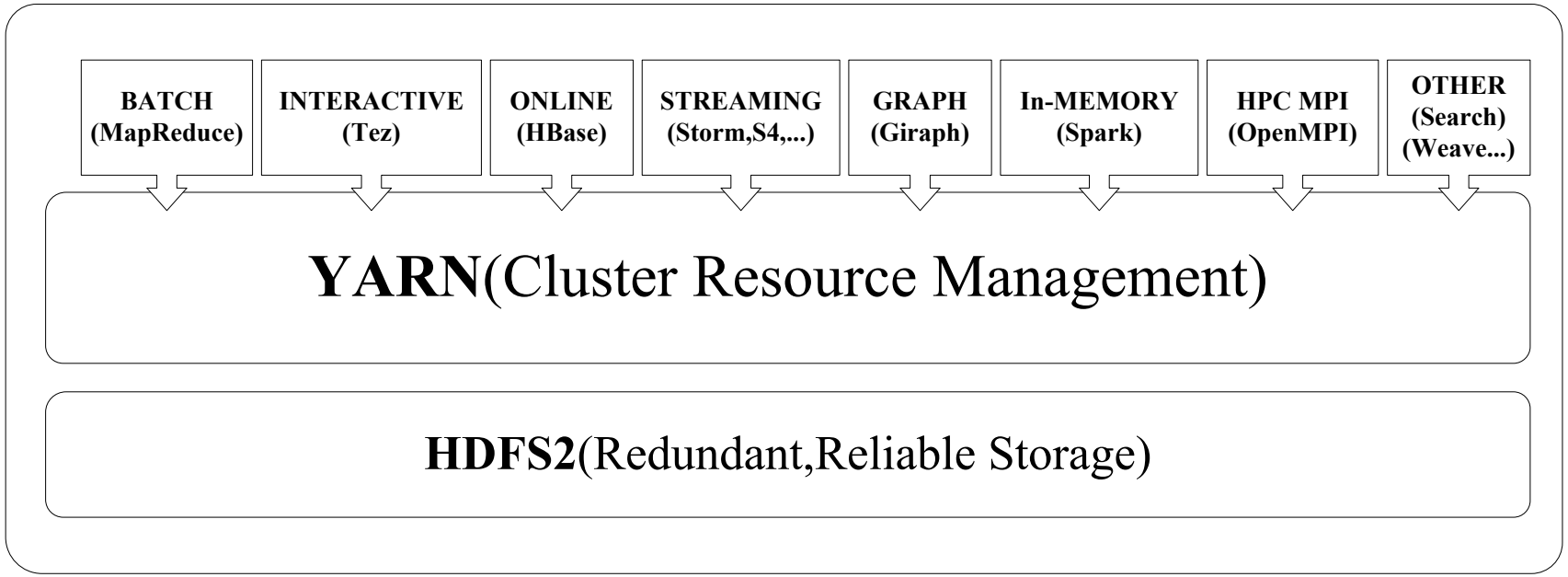
YARN Architecture （v2.0）



YARN

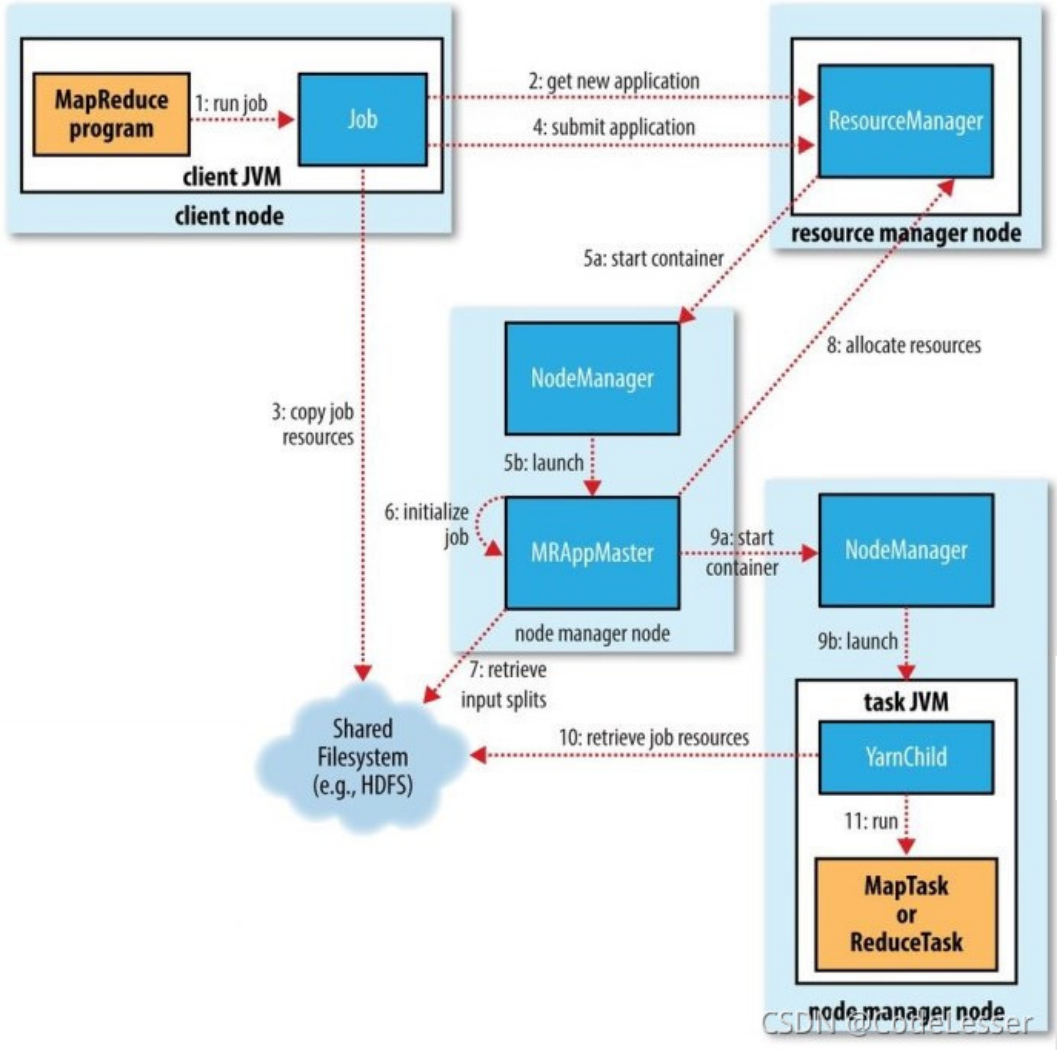
- YARN的目标就是实现“一个集群多个框架”，即在一个集群上部署一个统一的资源调度管理框架YARN，在YARN之上可以部署其他各种计算框架。
- 由YARN为这些计算框架提供统一的资源调度管理服务，并且能够根据各种计算框架的负载需求，调整各自占用的资源，实现集群资源共享和资源弹性收缩。
- 可以实现一个集群上的不同应用负载混搭，有效提高了集群的利用率。
- 不同计算框架可以共享底层存储，避免了数据集跨集群移动。

YARN



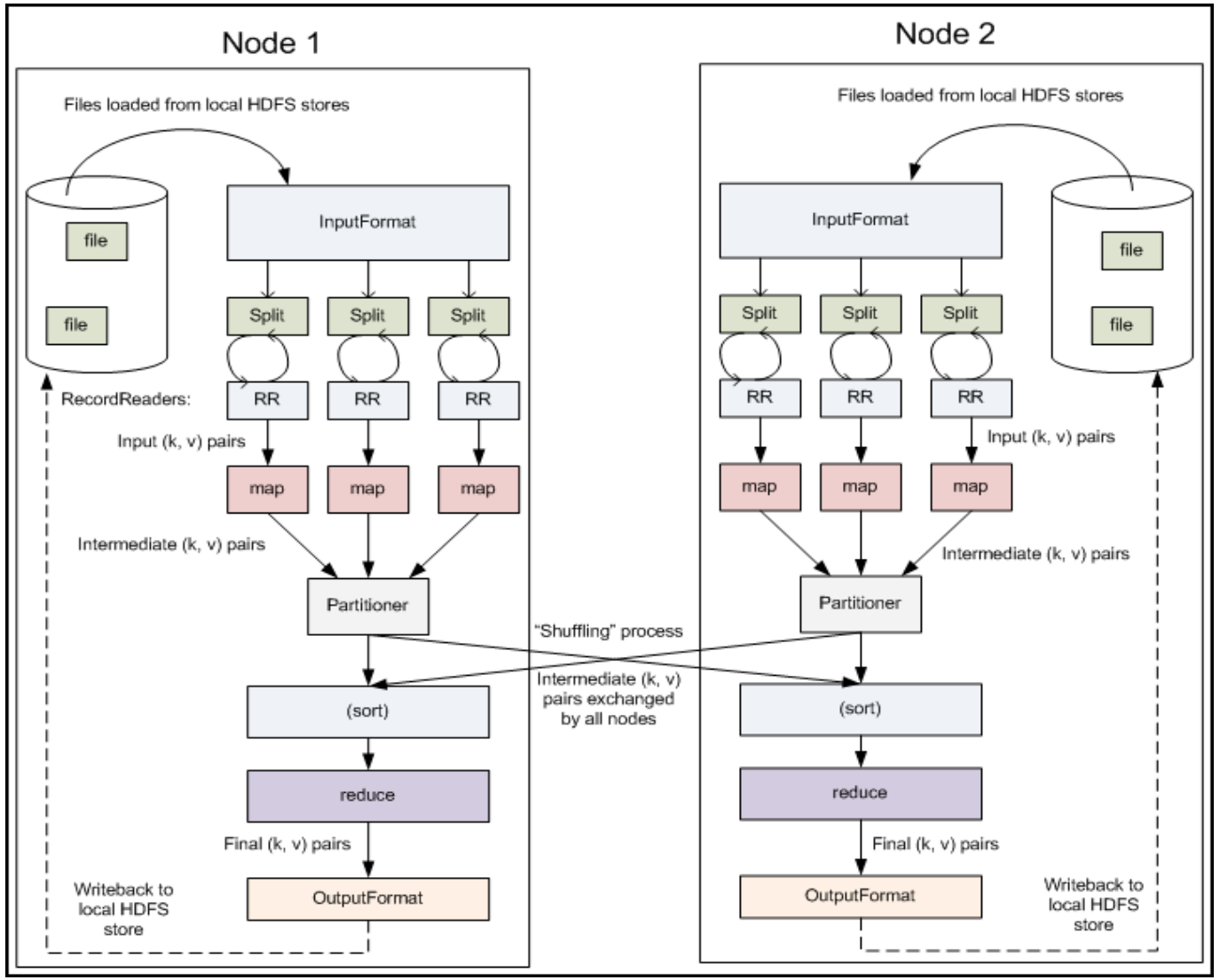
在YARN上部署各种计算框架

基于YARN的MapReduce架构（v2.0）



CSDN @CodeLesser

Hadoop MapReduce基本工作过程



THANK YOU

南京大學
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University