分布式文件系统HDFS

Grissom | 2025年10月

TRANSWARP
星 环 科 技

TRANSWARP
星 环 科 技

目录》
CONTENTS

1
chapter

HDFS简介

- ✓ HDFS概述
- ✓ 优缺点

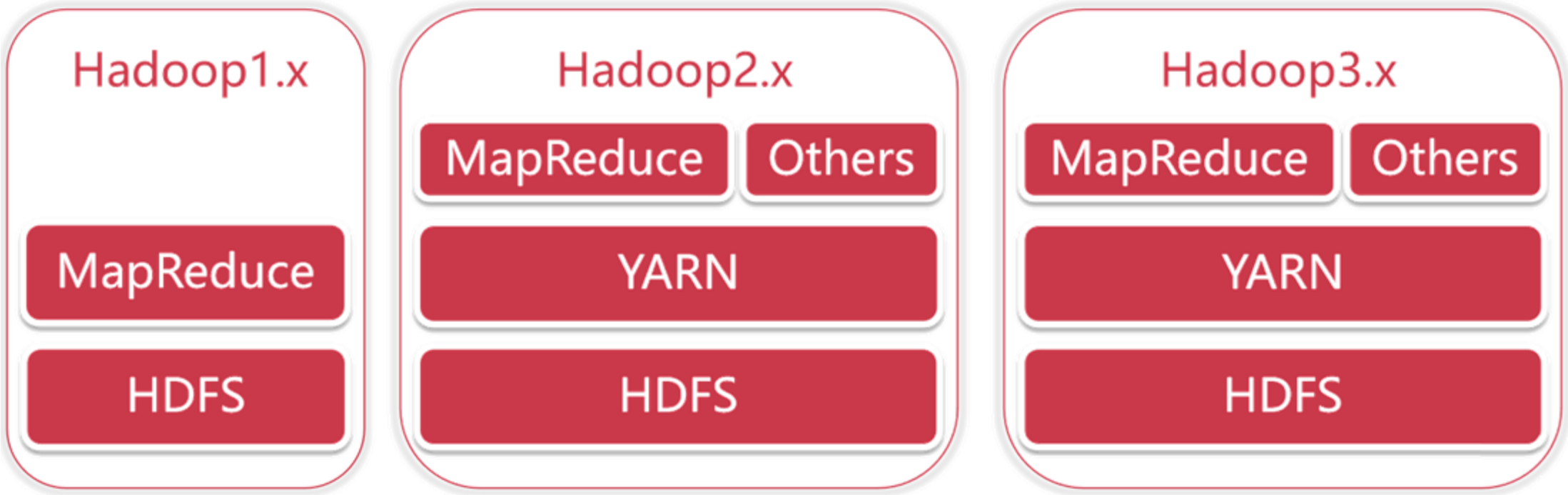
➢ 概念

- Hadoop分布式文件系统（Hadoop Distributed File System）
- 2003年10月Google发表了GFS（Google File System）论文
- HDFS是GFS的开源实现
- HDFS是Apache Hadoop的核心子项目
- 在开源大数据技术体系中，地位无可替代

➢ 设计目标

- 运行在大量廉价商用机器上：硬件错误是常态，提供容错机制
- 简单一致性模型：一次写入多次读取，支持追加写，但不允许并发写和随机修改，通过对写操作的严格限制来保证数据的一致性
- 流式数据访问：批量读而非随机读，关注吞吐量而非时间
- 存储大规模数据集：典型文件大小GB~TB（大文件），关注横向线性扩展

Hadoop版本演进



1.x -> 2.x

- MapReduce改进，新增YARN
- HDFS Namenode HA
- HDFS Federation

2.x -> 3.x

- 整体技术架构保持一致
- HDFS支持纠删码ErasuredCode
- HDFS支持多个Namenode，一主多备

1.2 优缺点

➢ 优点

- 高容错、高可用、高扩展
  - 数据冗余，多Block多副本，副本丢失后自动恢复
  - NameNode HA、安全模式
  - 10K节点规模
- 海量数据存储
  - 典型文件大小GB~TB，百万以上文件数量， PB以上数据规模
- 构建成本低、安全可靠
  - 构建在廉价商用服务器上
  - 提供了容错和恢复机制
- 适合大规模离线批处理
  - 流式数据访问
  - 数据位置暴露给计算框架

➢ 缺点

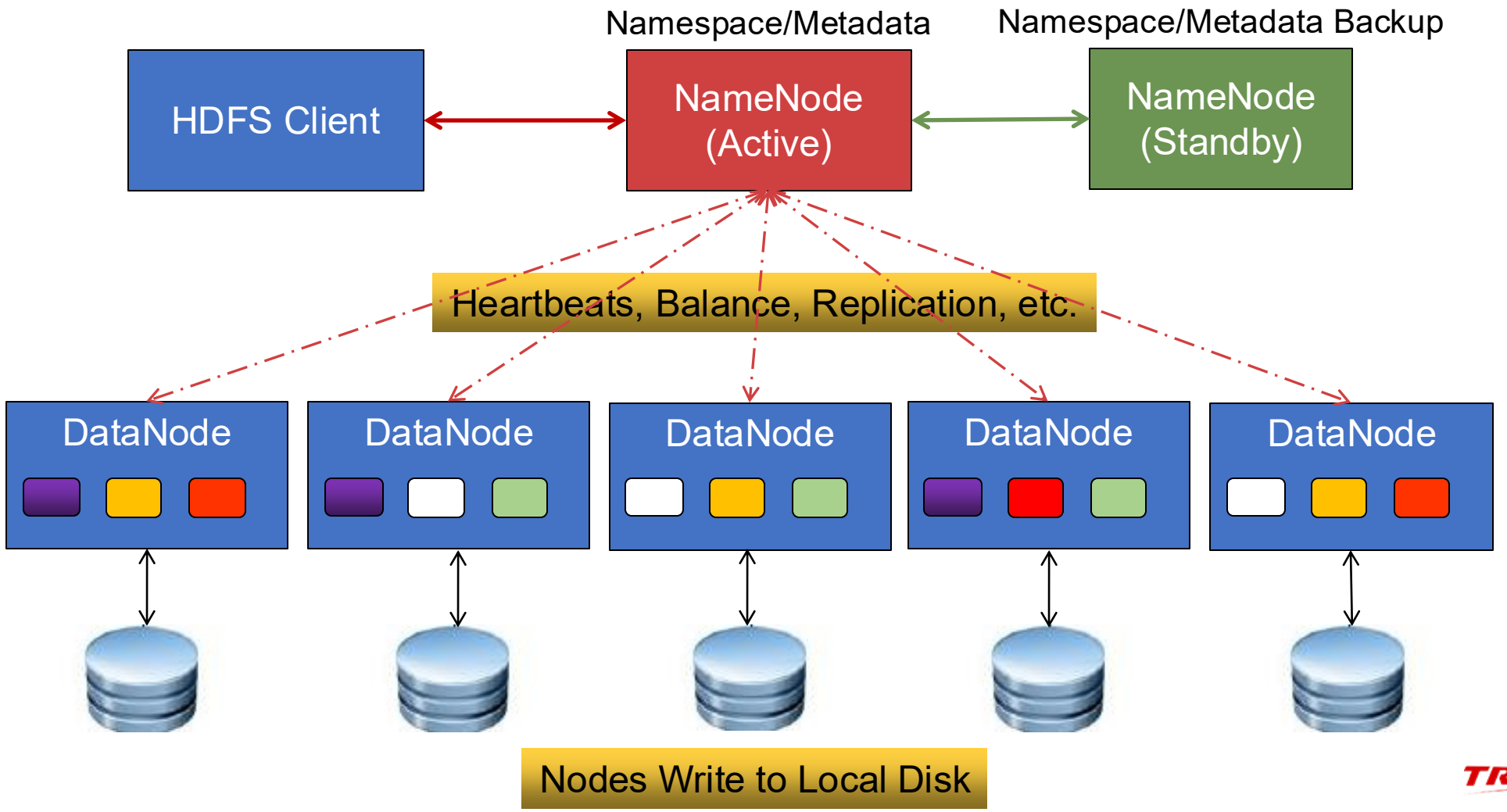
- 不适合低延迟数据访问
- 不适合存储小文件
  - 元数据占用NameNode大量内存空间
    - ✓ 每个文件或目录的元数据要占用150Byte
    - ✓ 存储1亿个文件，大约需要20GB内存
    - ✓ 如果一个文件为10KB，1亿个文件大小仅1TB，却要消耗掉20GB内存
  - 磁盘寻道时间超过读取时间
- 不支持并发写入
  - 一个文件同时只能有一个写入者
- 不支持随机修改
  - 仅支持追加写入

2
chapter

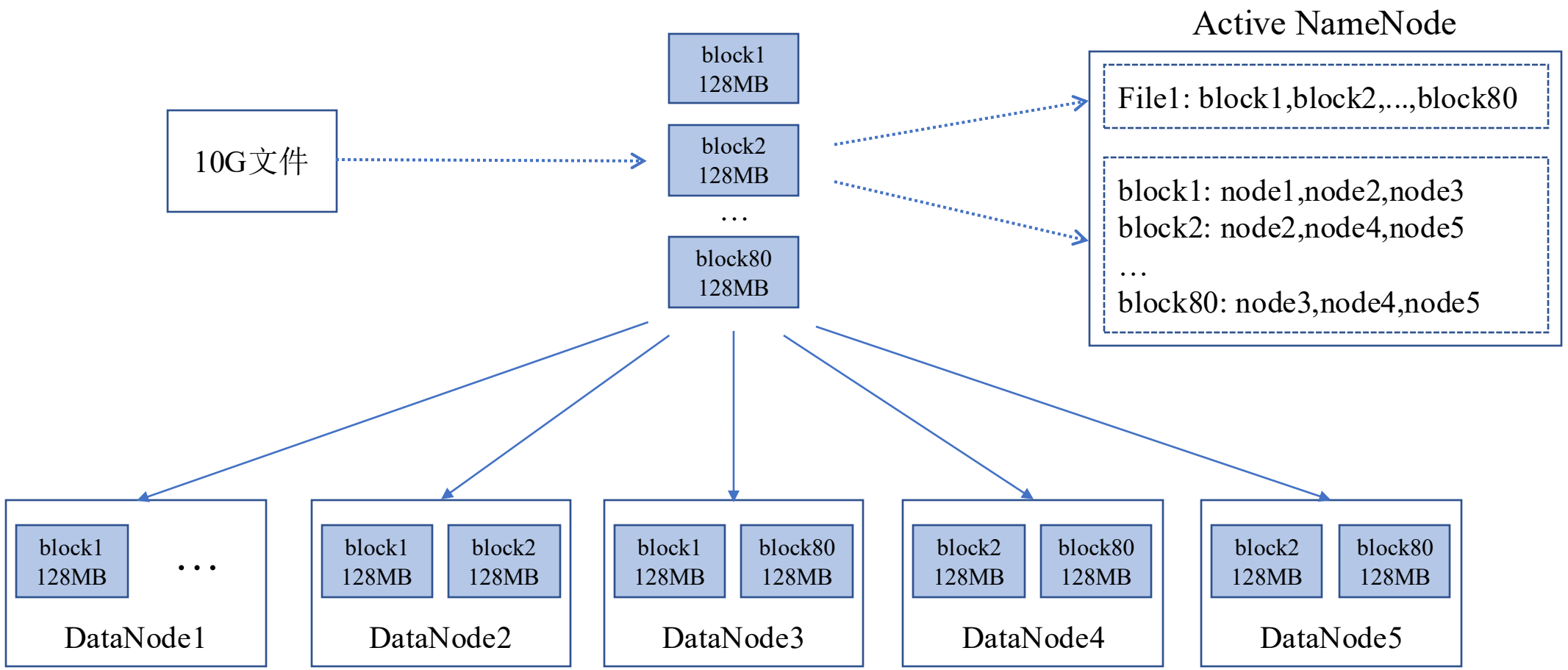
HDFS原理

- ✓ 系统架构
- ✓ 数据存储
- ✓ 读写操作
- ✓ 安全模式
- ✓ 高可用
- ✓ Federation
- ✓ 纠删码

- 系统架构图（Master/Slave + Active/Standby）



➢ 系统架构图（Master/Slave + Active/Standby）



2.1 系统架构：角色分工

- Active NameNode（AN）
  - 活动管理节点（Master / 集群中唯一）
  - 管理命名空间：文件系统的目录层次空间
  - 管理元数据：目录/文件的基本属性、Block相关信息、DataNode相关信息等
  - 管理Block副本策略：副本数（默认3副本）、副本放置策略等
  - 管理DataNode：节点上下线动态监测、文件系统健康状况监测、 Block恢复和重分布等
  - 处理客户端读写请求：审核权限，编制计划，并为DataNode分配任务
- Standby NameNode（SN）
  - 热备管理节点（允许多个）
    - Hadoop 3.0允许配置多个SN，之前的版本只能有一个
  - 主备切换：AN宕机后，经过Master选举和元数据恢复，SN升级为AN
  - 元数据同步：正常运行期间的周期性同步、AN宕机后的最新同步

2.1 系统架构：角色分工

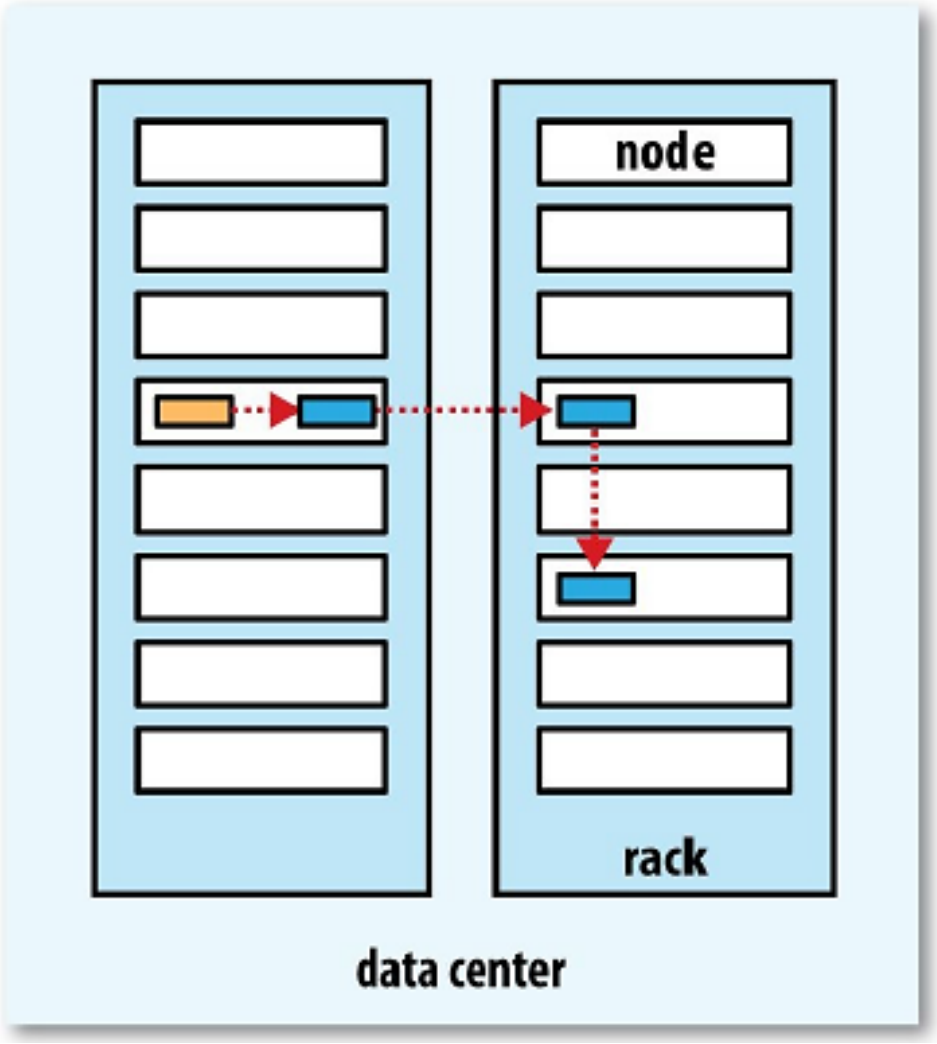
- DataNode（DN）
  - 数据节点（Slave / 高扩展）
  - 存储Block数据块和数据校验和
  - 向AN和SN汇报情况
    - 正常运行：通过心跳机制（默认3秒），定期汇报运行状况和Block信息（包括物理存储位置）
    - 集群启动：进入安全模式，集中上报Block信息
  - 执行客户端发送的读写命令
- Client
  - 文件管理
    - 发送文件读写请求，并将文件切分为Block或将Block组装为文件
    - 与NameNode交互，获取读写计划和相关元数据
    - 与DataNode交互，读取或写入Block
  - 系统管理：发送HDFS管理命令

➢ Block数据块

- HDFS的最小存储单元
- 多Block多副本
  - 文件被切分为若干个Block，每个Block有多个副本（默认3副本）
  - Block以DataNode为存储单元，即一个DataNode上只能存放Block的一个副本
  - 机架感知：尽量将副本存储到不同的机架上，以提升数据的容错能力
  - 副本均匀分布：DataNode的Block副本数和访问负荷要比较接近，以实现负载均衡
- Block大小
  - 默认128M（HDFS 1.x中，默认64M），可设置（若Block中数据的实际大小 < 设定值，则Block大小 = 实际数据大小）
  - 调整Block的大小
    - ✓ 目标：①最小化寻址开销，降到1%以下；②任务并发度和集群负载比较适中，作业运行速度较快
    - ✓ 块太小：①寻址时间占比过高；②任务太多，并发度太高，导致集群负载过高，作业变慢
    - ✓ 块太大：任务太少，并发度太低，导致集群负载过低，作业变慢

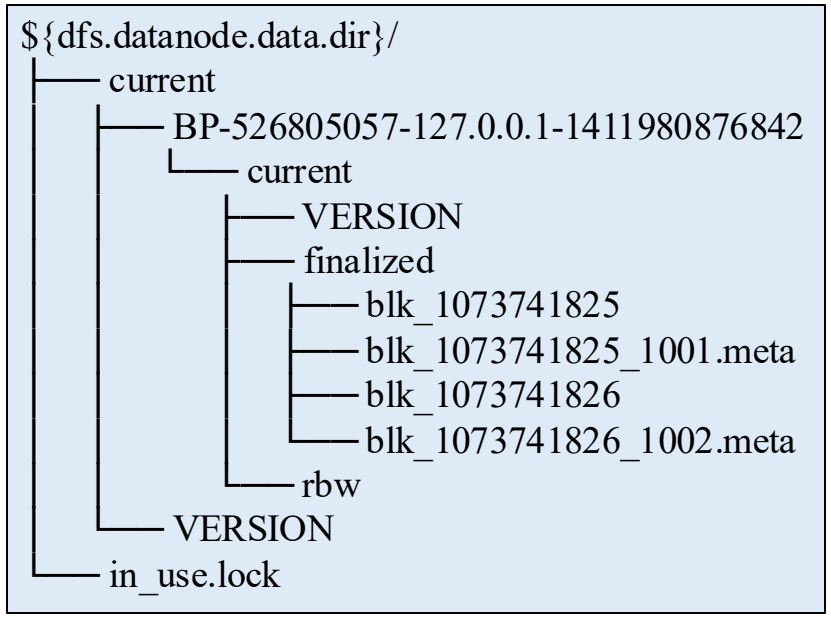
➢ Block副本放置策略（Hadoop 3.x）

- 副本1：放在Client所在节点上
  - 对于远程Client，系统会随机选择机架和节点
- 副本2：放在与副本1不同的机架上
- 副本3：放在与副本2同一机架的不同节点上
- 副本N：在遵循相关原则的前提下，随机选择
- 节点选择原则
  - 避免选择访问负荷太重的节点
  - 避免选择存储空间太满的节点
  - 将副本分散在不同的机架上



➢ Block文件

- Block数据文件：DataNode本地磁盘中名为“blk_blockId”的Linux文件
- Block元数据文件：DataNode本地磁盘中名为“blk_blockId_*.meta”的Linux文件，由一个包含版本、类型信息的头文件和一系列校验值组成
- Block文件目录：DataNode启动时自动创建，无需格式化



注：in_use.lock表示DataNode正在对文件夹进行操作

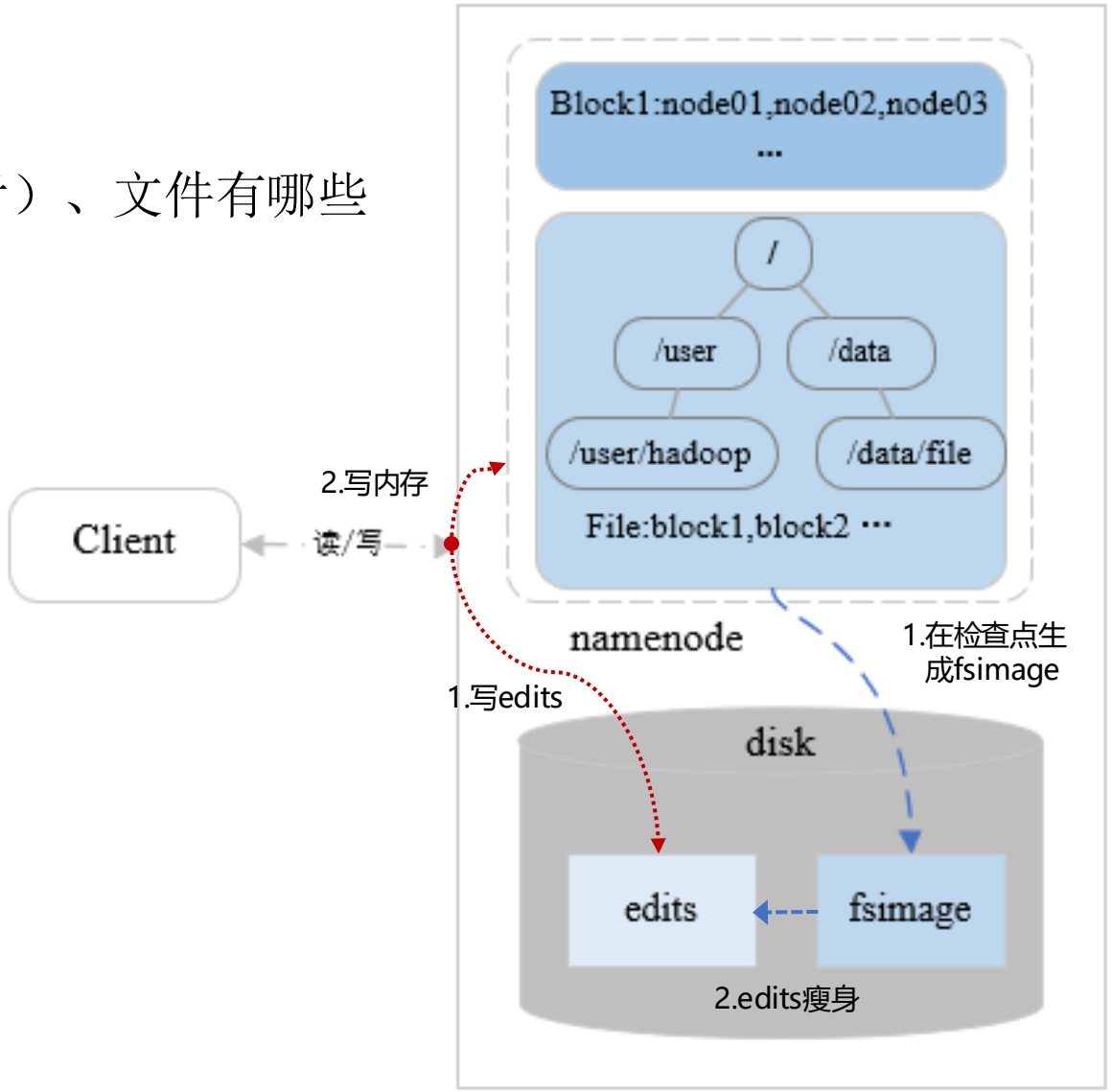
➢ 元数据

- 存放在NameNode内存当中
- 包含：目录/文件的基本属性（如名称、所有者）、文件有哪些block构成、以及block的位置存放信息。

➢ 元数据信息持久化

① fsimage（元数据检查点镜像文件）

② edits（编辑日志文件）



➢ 元数据信息持久化

```
${dfs.namenode.name.dir}/
├── current
│   ├── VERSION
│   ├── edits_0000000000000000001-0000000000000000019
│   ├── edits_inprogress_0000000000000000020
│   ├── fsimage_0000000000000000001
│   ├── fsimage_0000000000000000001.md5
│   ├── fsimage_0000000000000000019
│   ├── fsimage_0000000000000000019.md5
│   └── seen_txid
└── in_use.lock
```

注：in_use.lock表示NameNode正在对文件夹进行操作

- edits（编辑日志文件）
  - 记录文件系统的每一个变更操作，并定期瘦身
  - 先写edits，再写内存
  - edits文件名通过“txid前后缀”标记所包含变更操作的范围（txid为变更操作的Transaction id）
- fsimage（元数据检查点镜像文件）
  - 检查点（Checkpoint）：① 时间间隔（默认1小时），② 变更操作次数（默认100万）
  - 在检查点定期对内存中的元数据进行持久化，生成fsimage镜像文件（速度较慢）
  - fsimage文件名标记出最后一个变更操作的txid，以下图为例，只要在内存中载入fsimage_19，然后执行edits_inprogress_20，即可还原出最新的元数据
  - 在检查点，当fsimage生成后，删除“edits last txid < (fsimage last txid – txns)”的edits旧文件，从而实现定期瘦身（txns默认为100万）

➢ 文件元数据

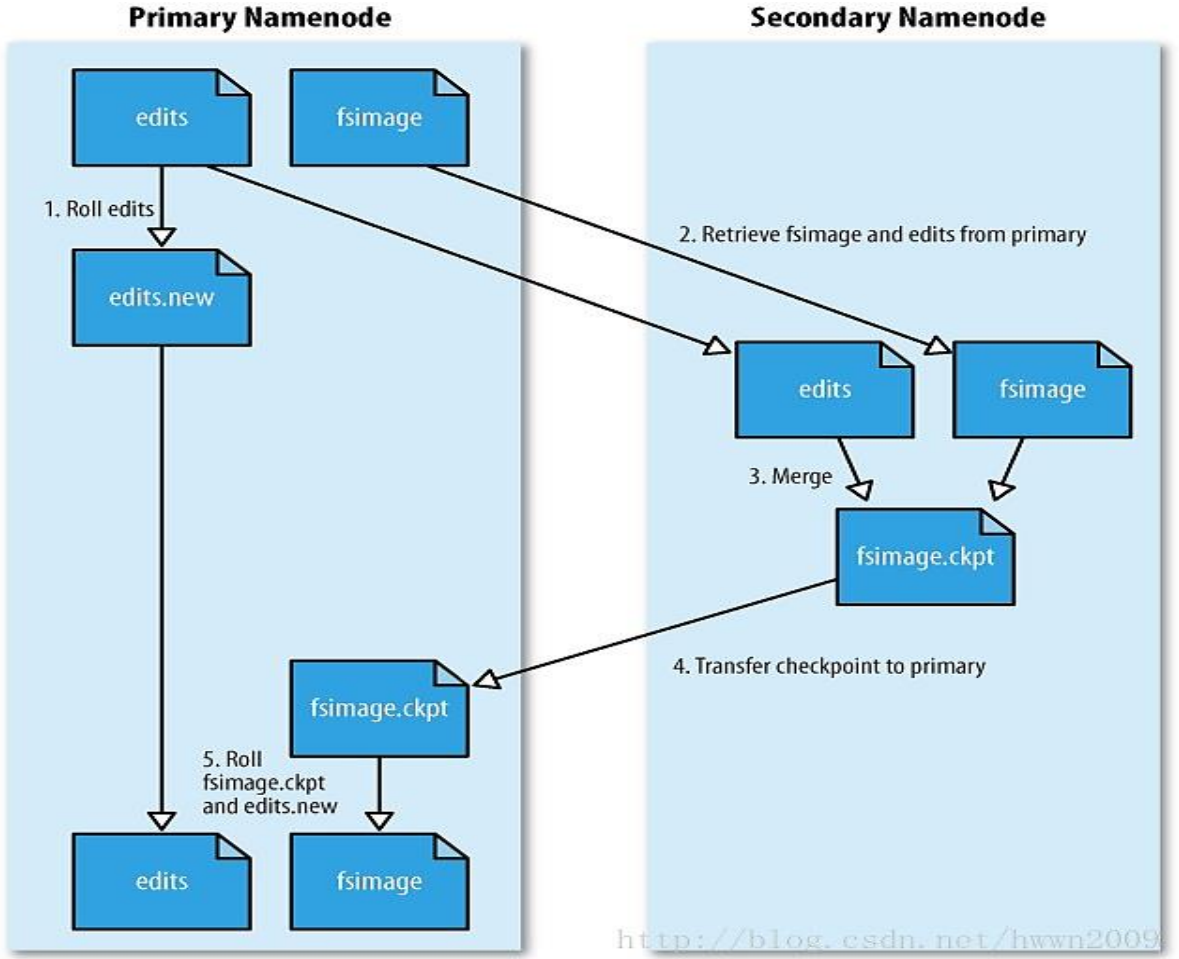
- edits与fsimage持久化（Hadoop 1.x）
  - 基于远程合并的持久化

    ① 在检查点，SN请求PN停用edits，后续变更操作写入edits.new

    ② 将fsimage和edits下载到SN（第一次需下载fsimage）

    ③ 在内存中载入fsimage，与edits进行合并，然后生成新的fsimage，并将其上传给PN

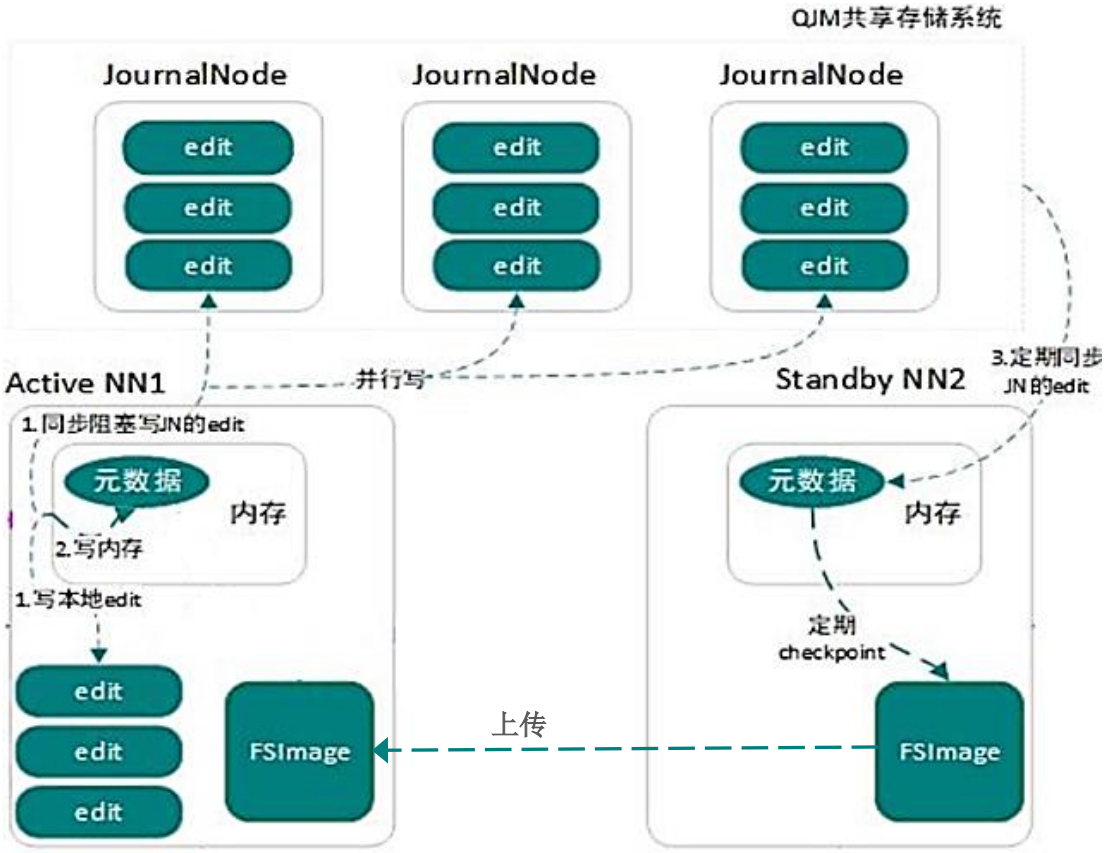
    ④ 用新文件替换edits和fsimage（edits实现瘦身）
  - 缺点
    - ✓ 合并前要先将fsimage载入内存，速度慢
    - ✓ 未实现edits高可用：SN上的edits不是最新的，若PN上的edits损毁，将导致元数据丢失



Hadoop 1.x

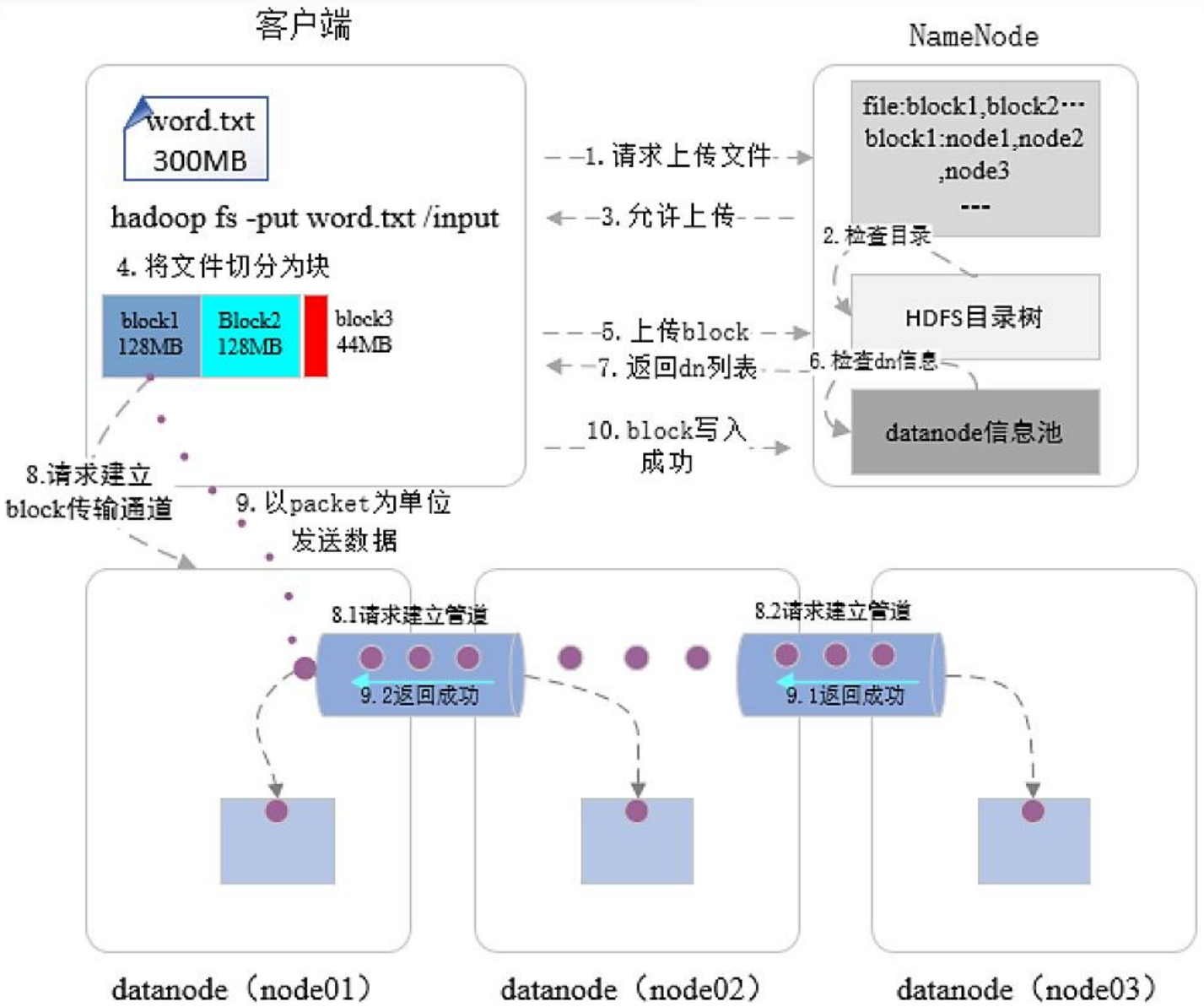
➢ 文件元数据

- edits与fsimage持久化（Hadoop 2.x）
  - QJM（Quorum Journal Manager）共享存储系统
    - ✓ 基于Paxos算法实现的JournalNode集群，实现了edits高可用和共享访问
    - ✓ 最好部署奇数（2*n*+1）个节点，最多容忍*n*个节点宕机
    - ✓ 过半（≥*n*+1）节点写入成功，即代表写操作完成
  - 基于QJM的edits持久化
    - ① AN将变更操作同步写入本地和QJM edits
    - ② AN将该操作写入内存，并将结果反馈给Client
  - 基于QJM的fsimage持久化
    - ① 在检查点，SN通过对内存中的NameSpace上锁，先将QJM edits定期同步（默认60s）暂停下来，再生成fsimage
    - ② SN将fsimage上传到AN，同时恢复QJM定期同步
    - ③ AN删除本地和QJM中的edits旧文件，完成瘦身



Hadoop 2.x

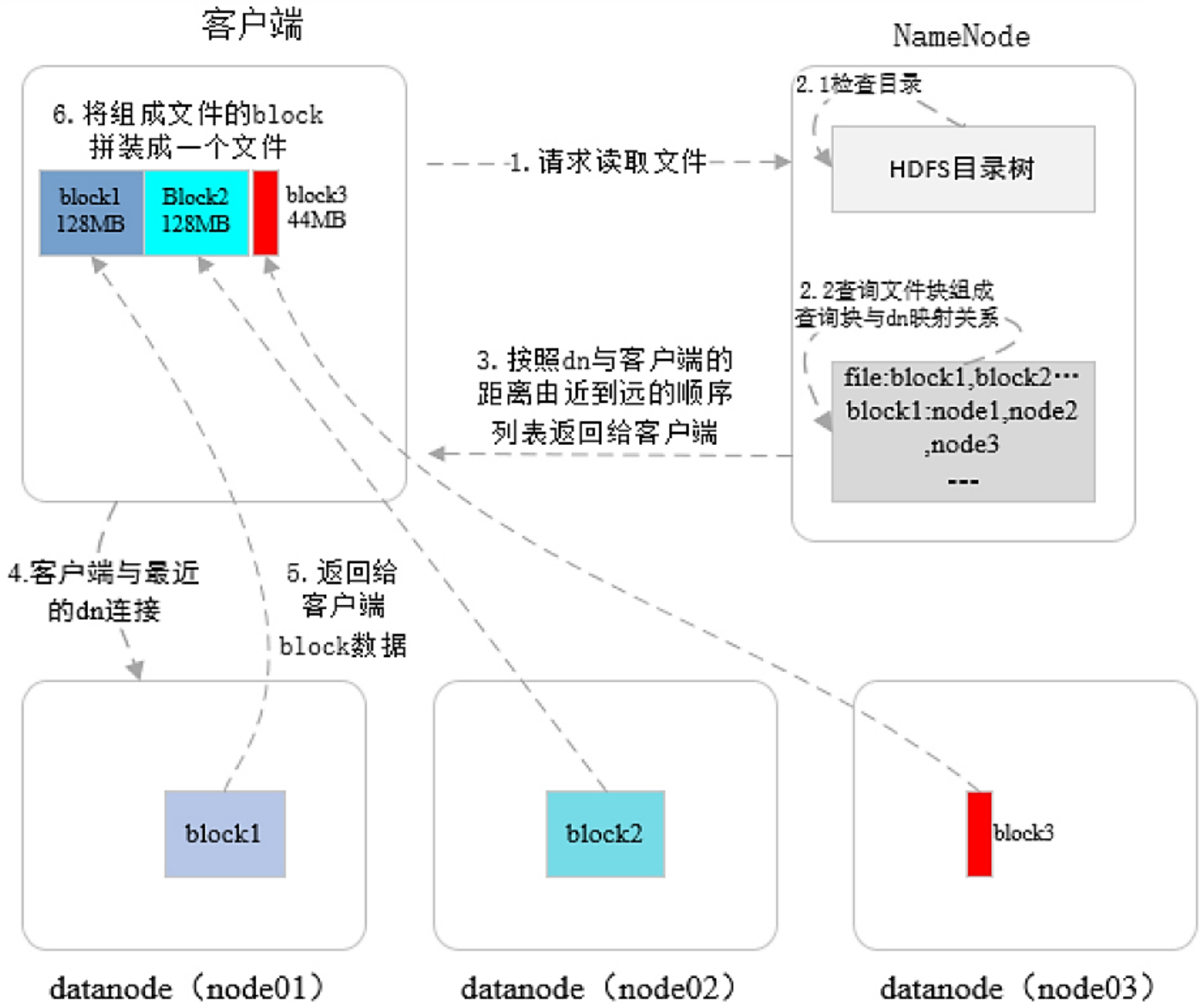
2.3 读写操作：写操作



➢ 写流程

- 客户端发送创建文件指令给分布式文件系统
- 文件系统告知namenode
- 检查权限，查看文件是否存在
- EditLog增加记录
- 返回输出流对象
- 客户端往输出流中写入数据，分成一个个数据包
- 根据namenode分配，输出流往datanode写数据
- 多个datanode构成一个管道pipeline，输出流写第一个，后面的转发
- 每个datanode写完一个块后，返回确认信息
- 写完数据，关闭输出流
- 发送完成信号给namenode

2.3 读写操作：读操作



➢ 读流程

- 客户端发送打开文件指令给分布式文件系统
- 文件系统访问namenode，获得这个文件的数据块位置列表，返回输入流对象
- 客户端从输入流中读取数据
- 输入流从各个datanode读取数据
- 关闭输入流

➢ 设计重点

- 客户端从datanode直接读取数据
- namenode只告知数据块的位置，不参与实际数据传输
- 高并发，namenode不成为瓶颈

➢ 什么是安全模式

- 安全模式是HDFS的一种特殊状态（只读），在该状态下HDFS只接收读请求，不接收写入、删除和修改等变更请求
- 安全模式是HDFS确保Block数据安全的一种保护机制
- Active NameNode启动时，HDFS会自动进入安全模式，DataNode主动向NameNode上报可用Block信息，在达到安全标准前，HDFS一直处于“只读”状态

➢ 何时离开安全模式

- Block上报率：DataNode上报的可用Block个数 ÷ NameNode元数据记录的Block个数
- 当Block上报率 >= 阈值时，HDFS才能离开安全模式（默认阈值为0.999）
- 不建议手动强制退出安全模式

➢ 触发安全模式的原因

- NameNode重启
- NameNode磁盘空间不足
- DataNode无法正常启动
- Block上报率低于阈值
- 日志中出现严重异常
- 用户操作不当，如强制关机（特别注意）

➢ 故障排查

- 找到DataNode不能正常启动的原因，重启DataNode
- 清理NameNode磁盘

➢ 背景：

- Hadoop1.x中，如果NameNode宕机，可能会出现数据丢失的问题，因为SecondaryNameNode上没有完整的、最新的元数据信息
- Hadoop2.x中，对于HDFS中NameNode高可用提供了实现

➢ 实现思路，思考：

1. 集群中，提供两台NameNode做热备

   Active + Standby

2. 一台NameNode宕机，另外一台如何接管集群?

   a) 元数据信息保持一致

      edits

      fsimage

   b) 如何做到自动的故障迁移?

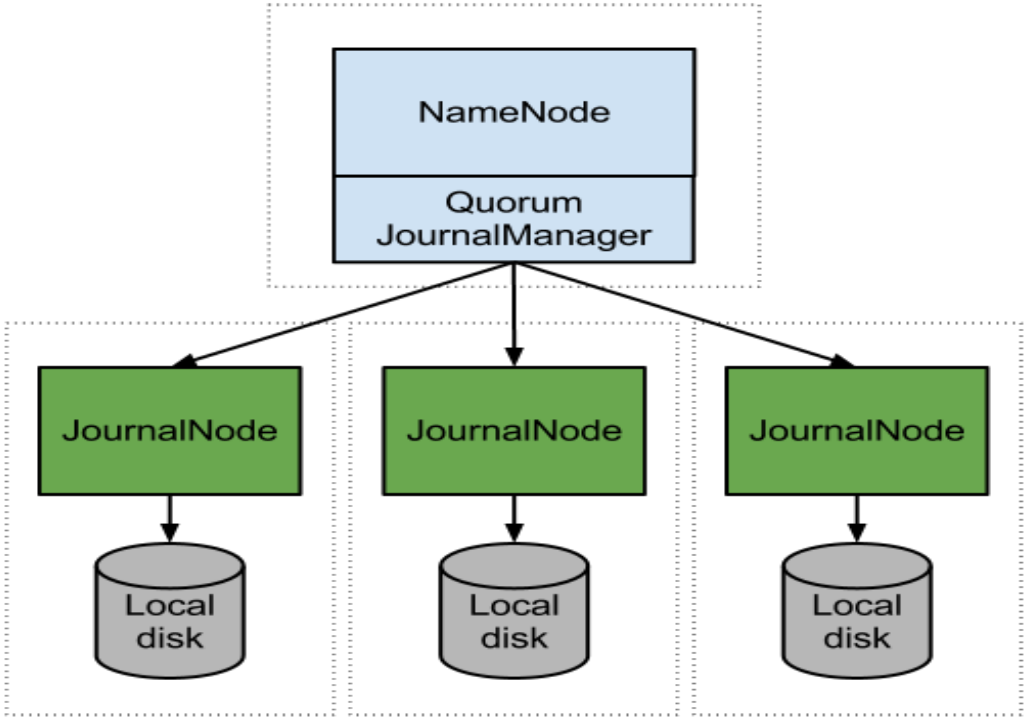
      Zookeeper集群

Journal集群

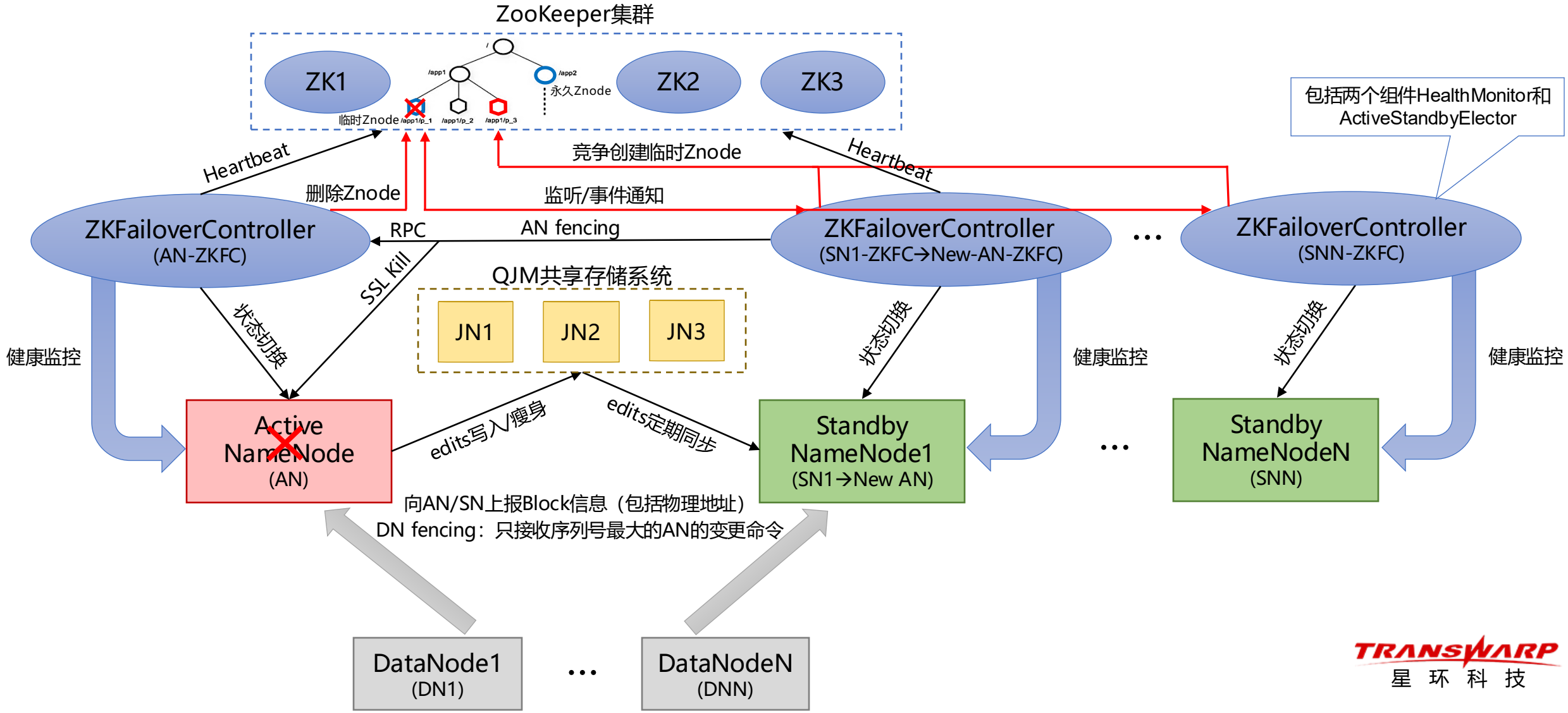
- 负责存储edits编辑日志
- 部署奇数（2N+1）台服务器

利用QJM实现元数据edits文件高可用

- QJM机制（Quorum Journal Manager）
  - 只要保证Quorum（法定人数）数量的操作成功，就认为这是一次最终成功的操作
- QJM共享存储系统
  - 写edits的时候，只要超过半数（>=N+1）的JournalNode返回成功，就代表本次写入成功
  - 最多可容忍N个JournalNode宕机
  - 基于Paxos算法实现



➢ NameNode高可用（3.x，一主多备）



➢ NameNode高可用（主备切换）

- 基于ZK实现Master选举

（1）AN或AN-ZKFC宕机，其创建的临时Znode被删除，但永久Znode（存储AN地址）被保留

– 若AN宕机，ZKFC失去与AN的心跳连接，从而监测到故障，然后主动删除临时Znode

– 若AN-ZKFC宕机，ZK失去与ZKFC的心跳连接，会话超时后，临时Znode被自动删除

（2）Watcher触发，通知所有SN-ZKFC去竞争创建临时Znode，SN1-ZKFC创建成功，SN1被选举为New AN

（3）其他SN-ZKFC在临时Znode上注册监听器，等待下一次选举

- 通过AN-Fencing防止脑裂

（4）若New-AN-ZKFC发现永久Znode，则通过RPC调用AN-ZKFC的服务，将AN从Active切换为Standby

（5）若步骤④失败，则New-AN-ZKFC直接调用SSL命令（或者执行自定义Shell脚本），Kill AN进程

（6）若步骤④⑤均失败，则需要人工干预

（7）AN-Fencing成功后，New-AN-ZKFC将New AN从Standby切为Active，并正式对外服务（HDFS只读）

- 基于QJM实现元数据恢复

（8）New AN从QJM edits中下载最新的（≤60s）变更操作，并在内存中执行一遍，即可恢复

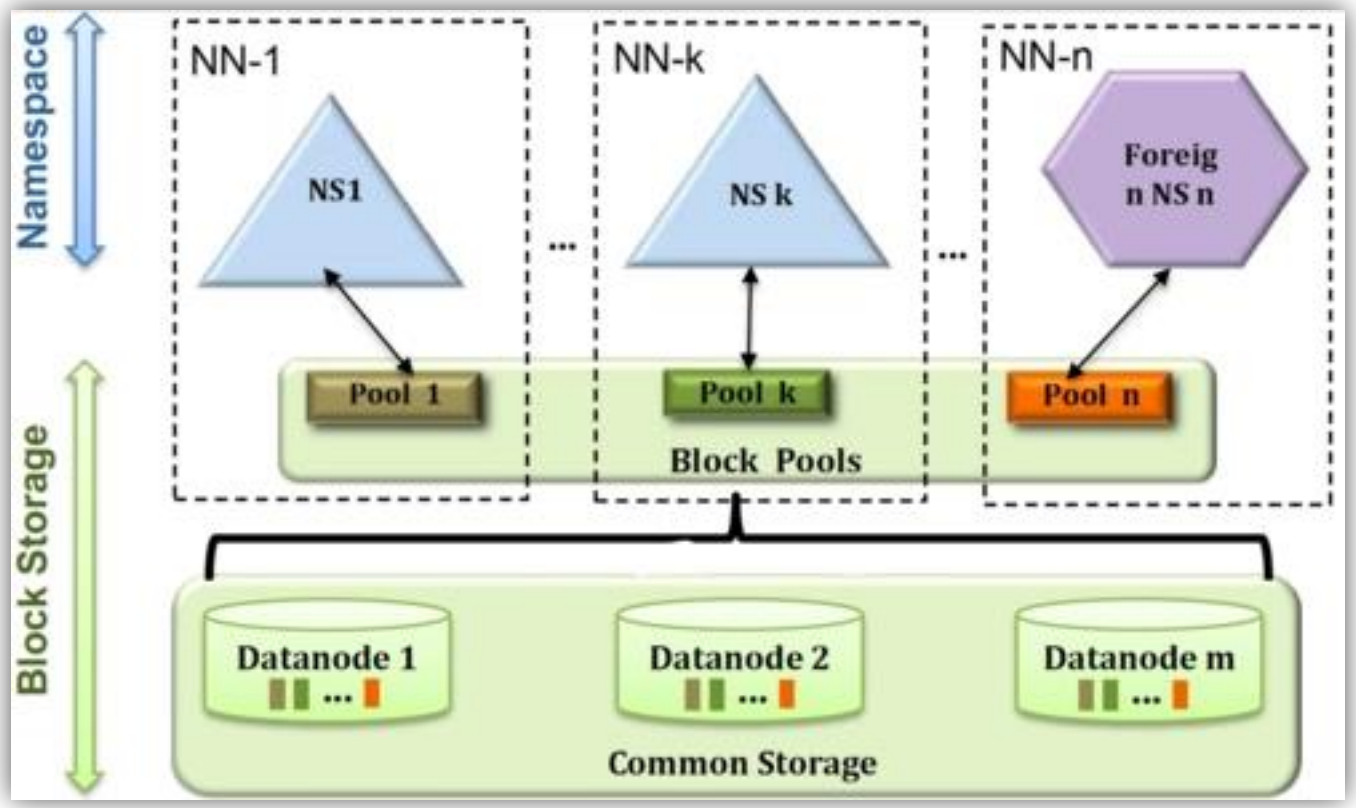
2.6 Federation

➢ 背景：

- Hadoop1.x中，由单一的NameNode管理文件系统的元数据信息，随着数据量的增长，当NameNode内存不足以存储下所有的元数据，此时NameNode内存会出现瓶颈。
- Hadoop2.x中，通过Federation机制解决NameNode内存受限问题。

➢ 实现机制：

- Federation允许一个HDFS集群中存在多组Namenode同时对外提供服务，分管一部分目录（水平切分），彼此之间相互隔离，但共享底层的Datanode存储资源。



2.6 Federation

- 优势：
  - ✓ 扩展性

    通过水平切分命名空间，HDFS Federation允许集群在保持高性能的同时进行横向扩展。这使得HDFS能够支持更大规模的数据存储和处理。
  - ✓ 可用性

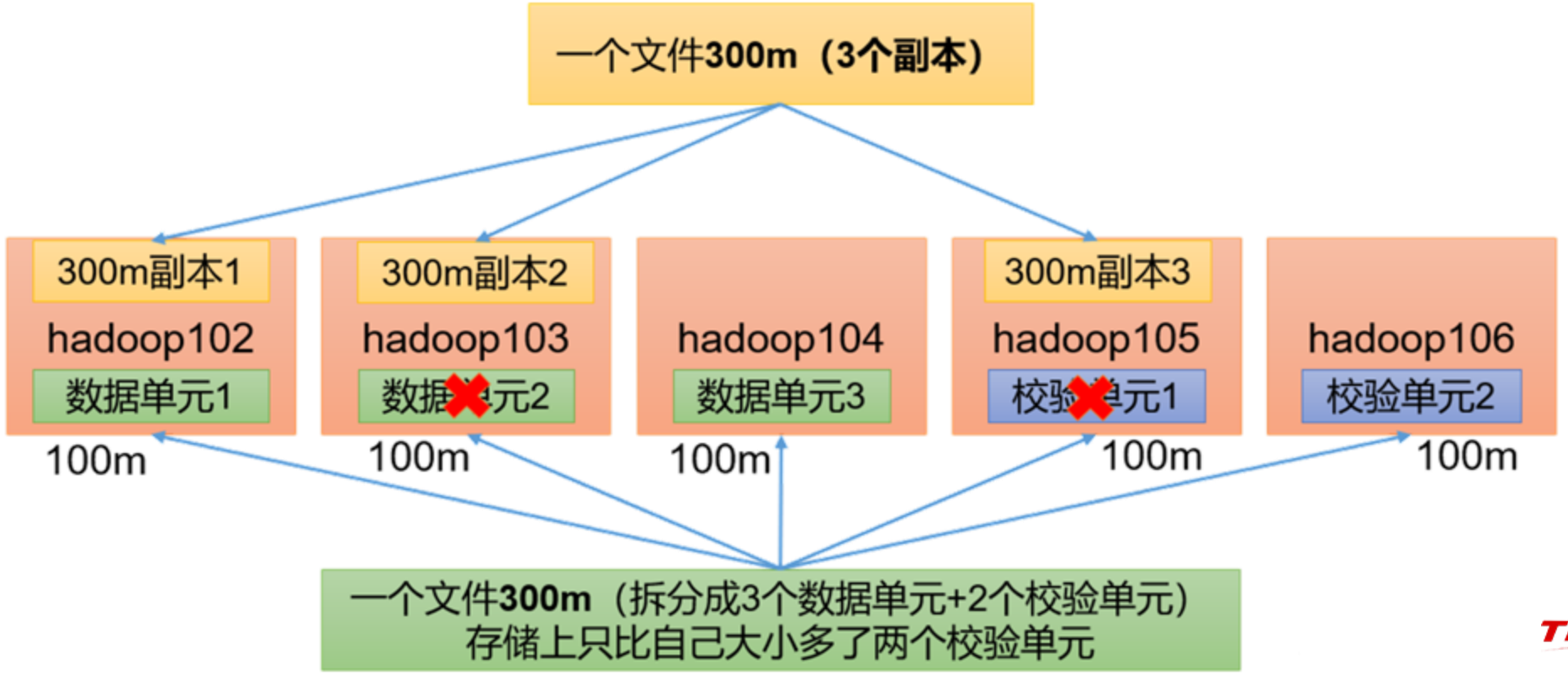
    在传统的HDFS架构中，单个Namenode的故障可能导致整个集群不可用。而HDFS Federation通过多个独立的Namenode分担任务，降低了单点故障的风险，提高了系统的可用性。
  - ✓ 负载均衡

    HDFS Federation中的多个Namenode可以根据负载情况动态调整分配的数据块，从而实现集群的负载均衡，提高了系统的性能。

- 适用场景：
  - 主要适用于大规模数据存储和处理的场景，特别是在需要高可用性、高性能和可扩展性的应用中。

➢ 纠删码ErasuredCode

- HDFS默认情况下，一个文件有3个副本，在提高数据可靠性的同事，带来了2倍的冗余开销。
- Hadoop3.x引入了纠删码，采用计算的方式，存储成本节省约50%左右。

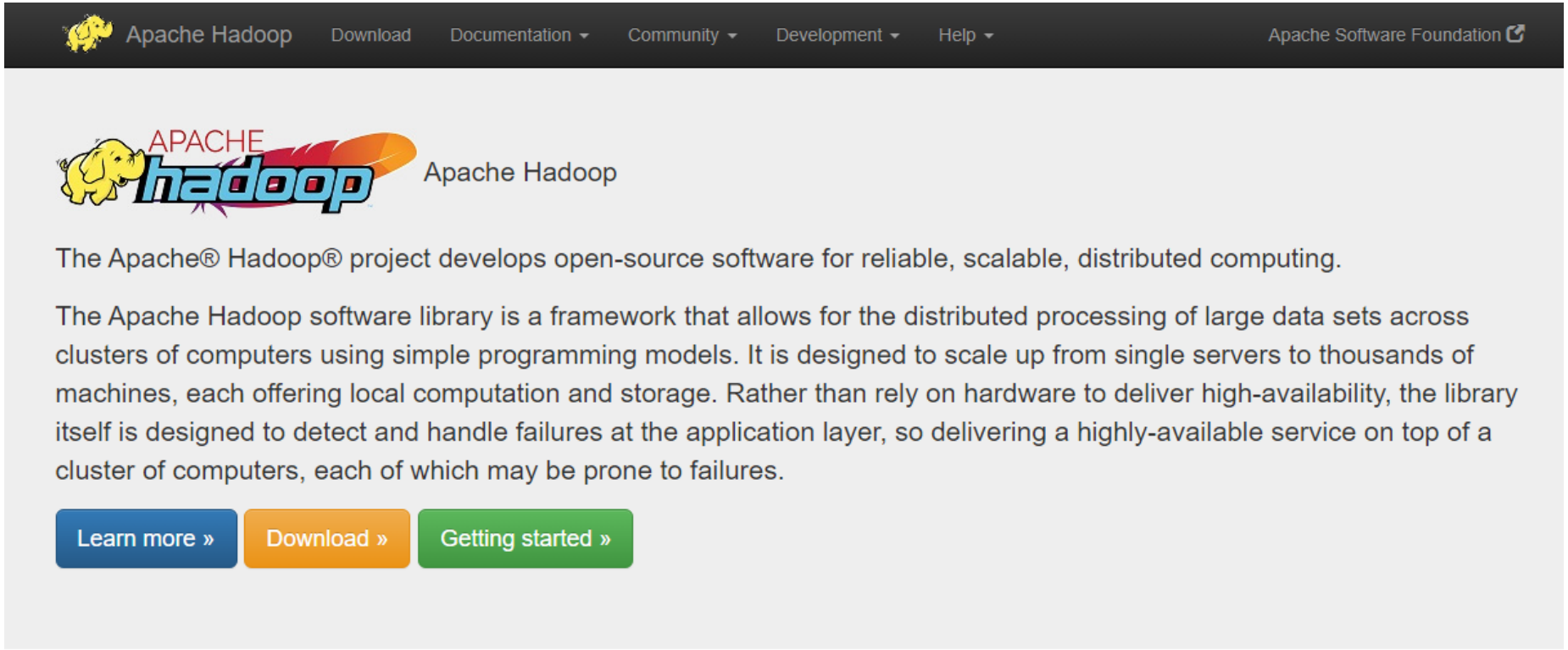


3
chapter

HDFS集群部署

- 官网地址：

https://hadoop.apache.org/



- 部署模式：
  - ① 伪分布式
    - 使用一台服务器（虚拟机）来模拟Hadoop集群
    - 一般用于开发、测试
  - ② 完全分布式
    - 一般用于开发、测试
  - ③ 高可用集群
    - 需提前部署Zookeeper集群

详细部署流程，可参照Hadoop官网 或Hadoop部署手册。

Q&A

TRANSWARP
星 环 科 技