COMP9313: Big Data Management

Hadoop and HDFS

Hadoop

- Apache Hadoop is an open-source software framework that

  分布式方式

  - Stores big data in a distributed manner
  - Processes big data parallelly 并行处理大数据

    建立在大型商品硬件集群上
  - Builds on large clusters of commodity hardware.
- Based on Google's papers on Google File System(2003) and MapReduce(2004).
- Hadoop is

  扩展 体积
  - Scalable to Petabytes or more easily (Volume)

    提供并行数据处理
  - Offering parallel data processing (Velocity) 速度
  - Storing all kinds of data (Variety)

Hadoop offers

冗余，容错数据存储

- Redundant, Fault-tolerant data storage (HDFS)

并行计算框架

- Parallel computation framework (MapReduce)

工作协调/安排

- Job coordination/scheduling (YARN)

- Programmers no longer need to worry about
  - Where file is located?
  - How to handle failures & data lost?
  - How to divide computation? 除法计算
  - How to program for scaling? 缩放程序

生态系统

Hadoop Ecosystem

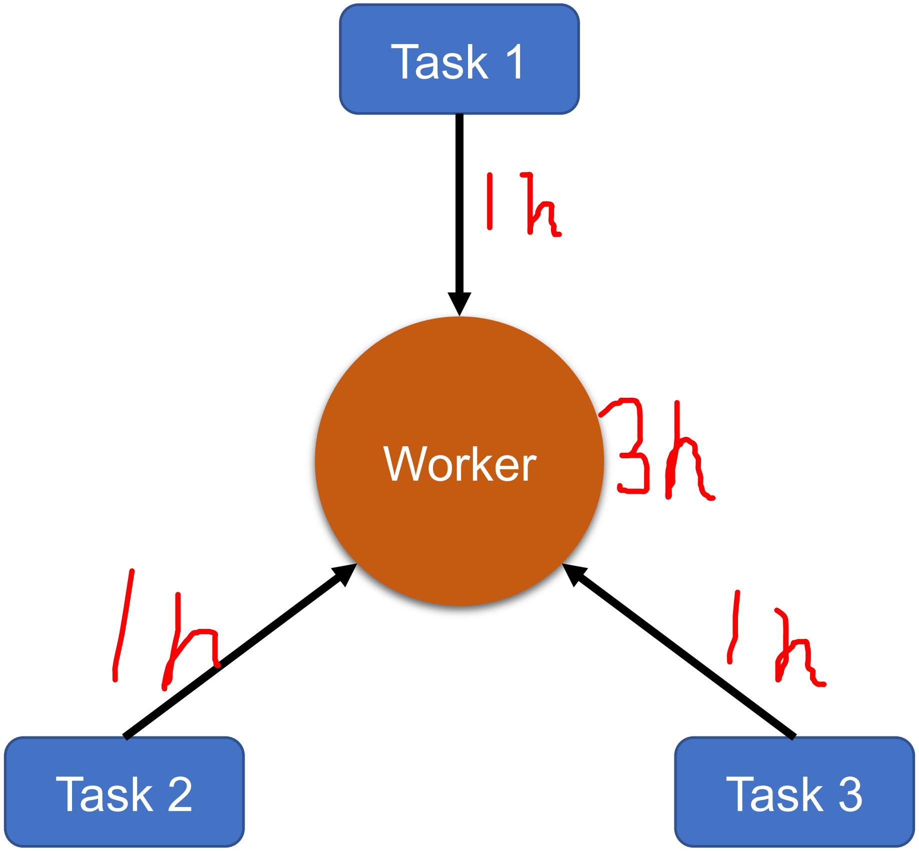
Hadoop的核心

- Core of Hadoop
  - Hadoop distributed file system (HDFS)
  - MapReduce
  - 另一个资源谈判者
  - YARN (Yet Another Resource Negotiator) (from Hadoop v2.0)
- Additional software packages
  - Pig
  - Hive
  - Spark
  - HBase
  - …

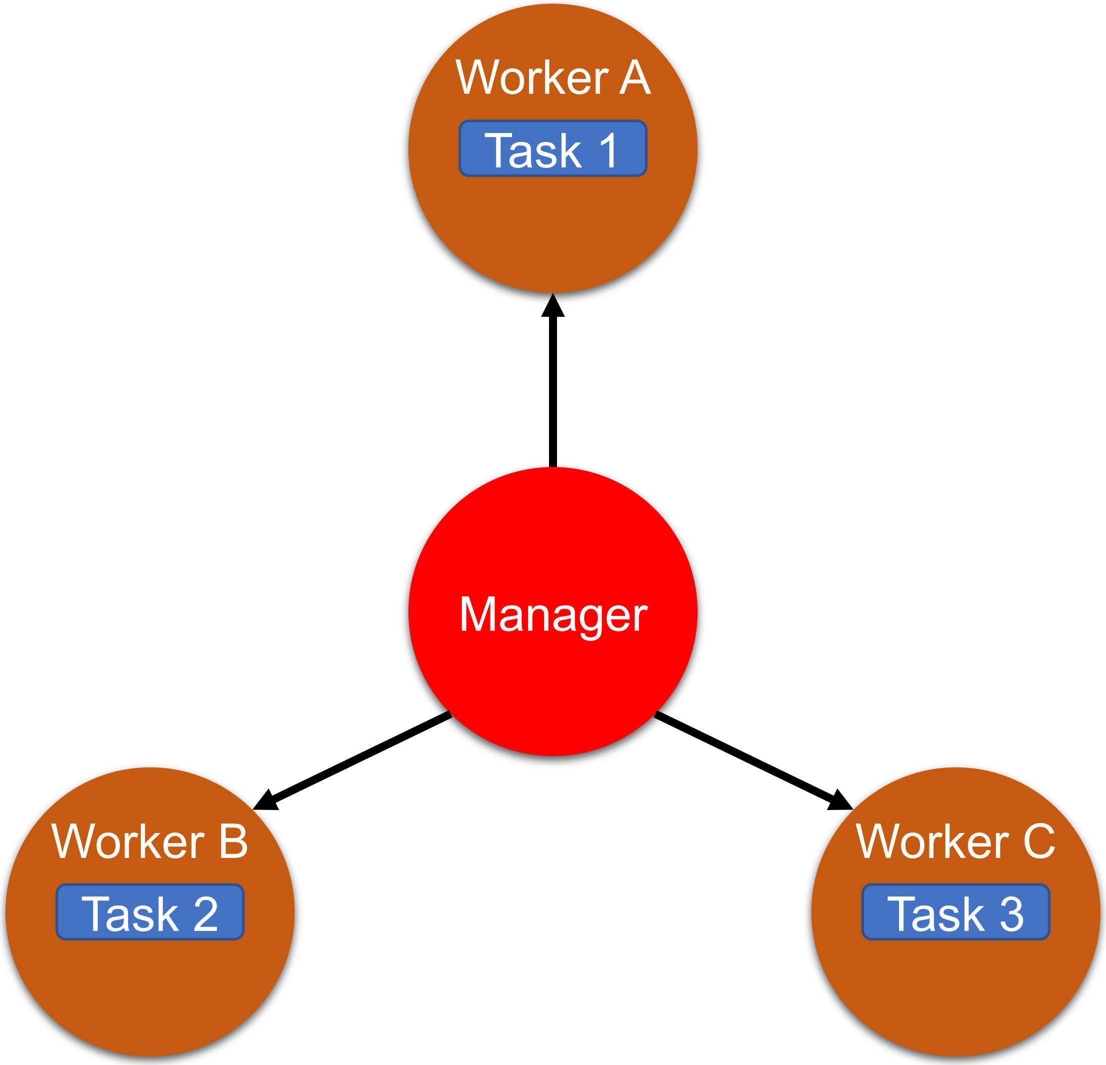
Hadoop是一个由Apache基金会所开发的分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），简称HDFS。HDFS有高容错性的特点，并且设计用来部署在低廉的（low-cost）硬件上；而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。HDFS放宽了（relax）POSIX的要求，可以以流的形式访问（streaming access）文件系统中的数据。Hadoop的框架最核心的设计就是：HDFS和MapReduce。HDFS为海量的数据提供了存储，而MapReduce则为海量的数据提供了计算

Hadoop主从架构

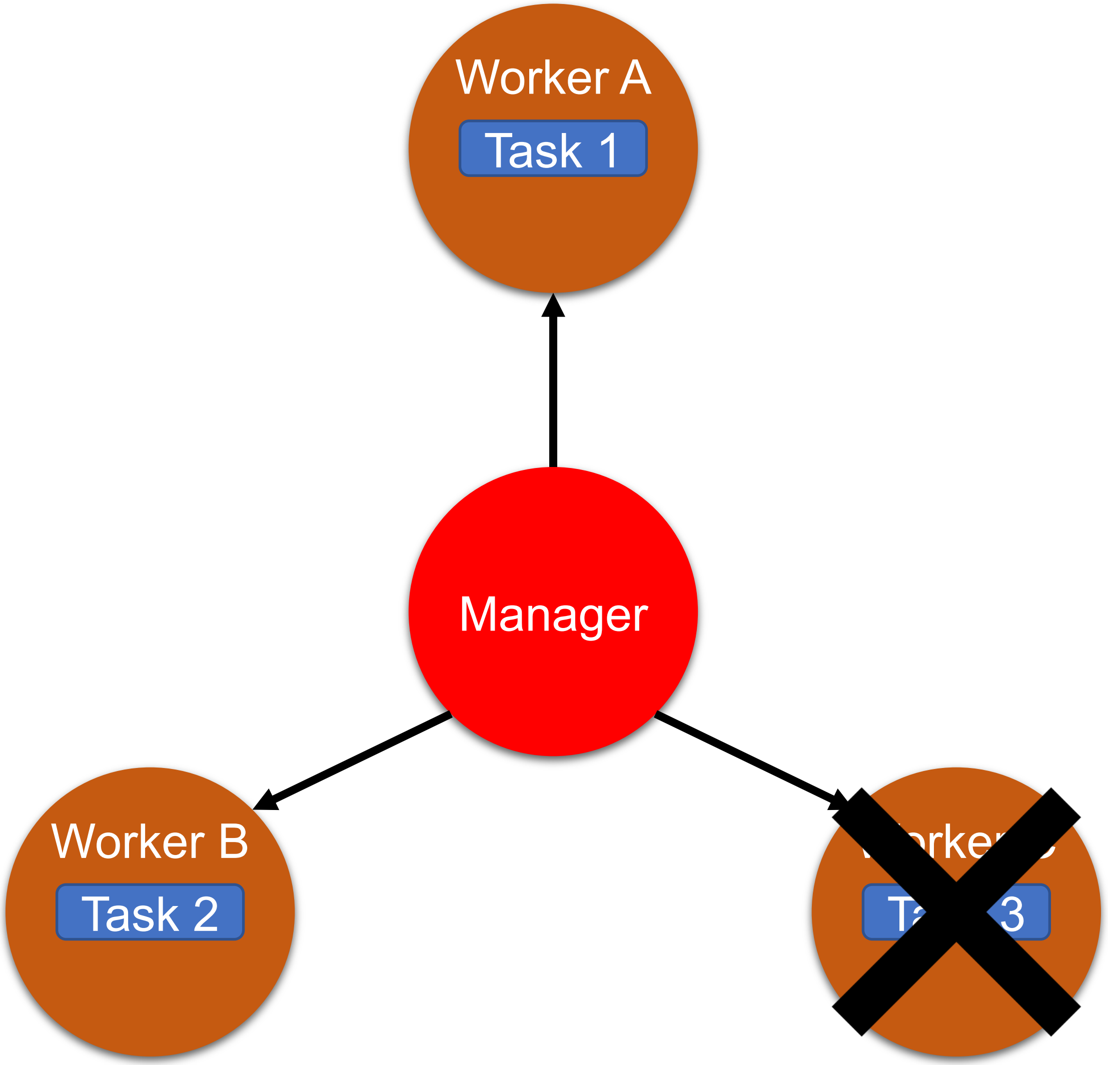
The Master-Slave Architecture of Hadoop



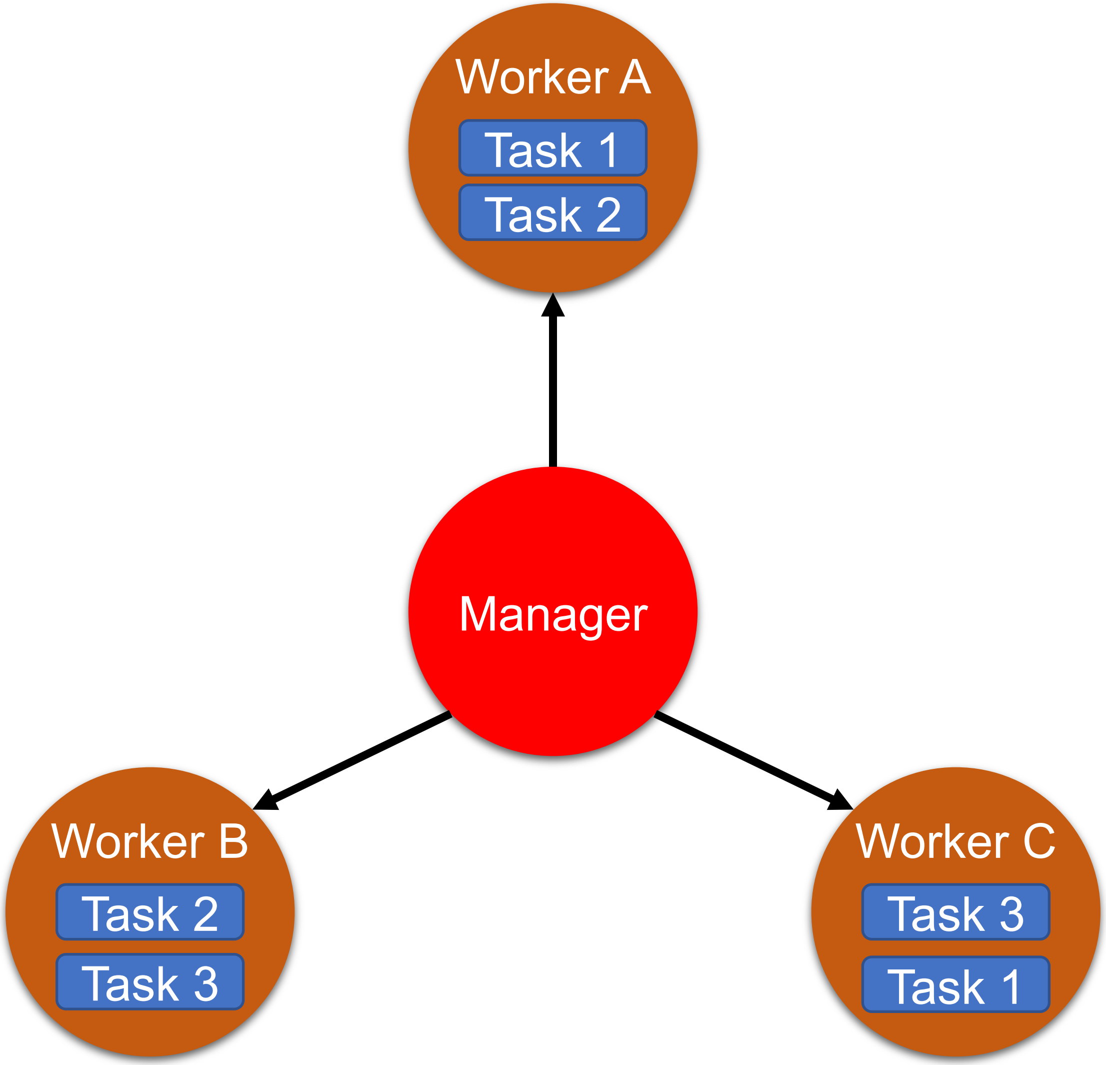
The Master-Slave Architecture of Hadoop



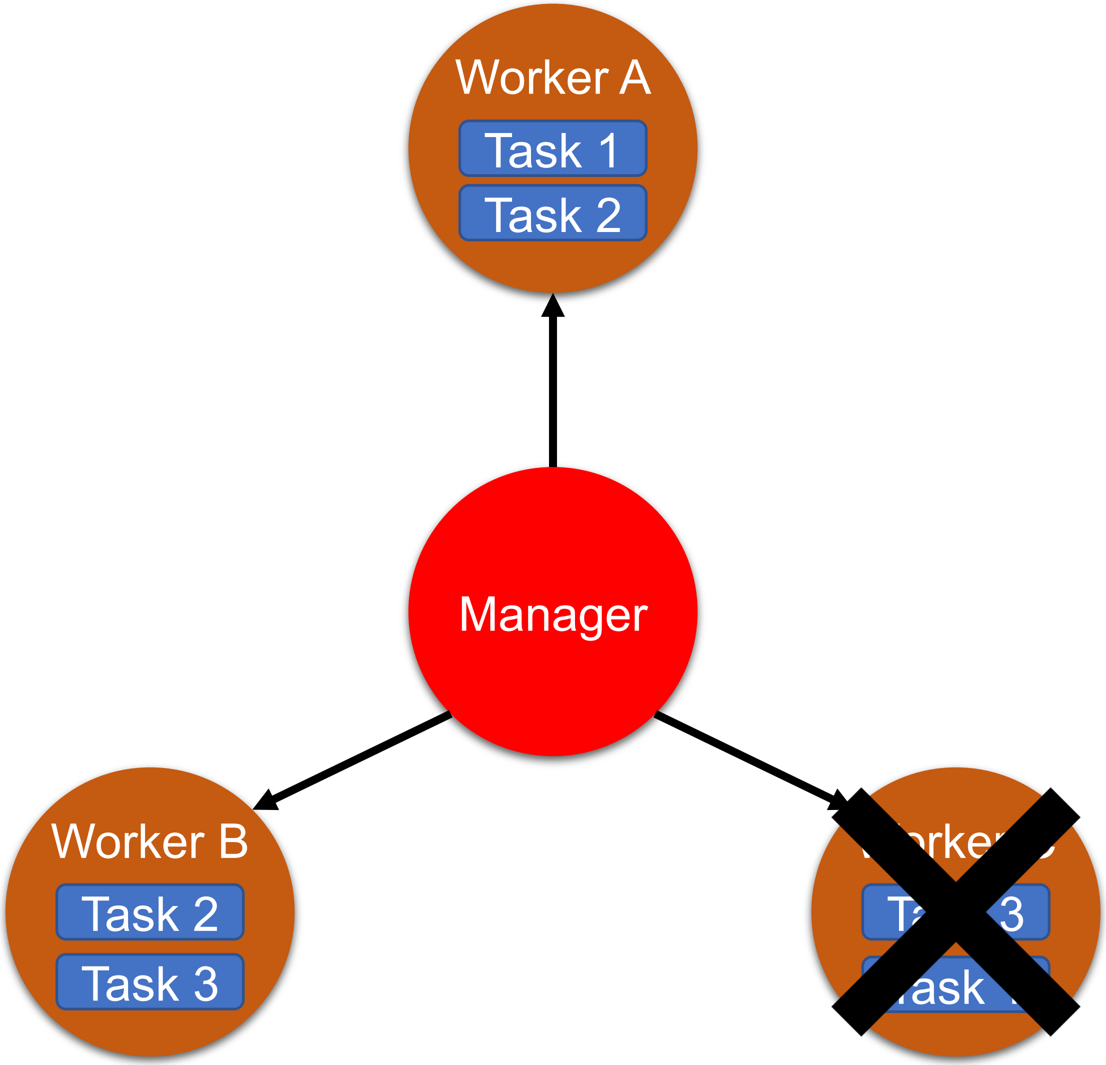
The Master-Slave Architecture of Hadoop



The Master-Slave Architecture of Hadoop



The Master-Slave Architecture of Hadoop



Hadoop分布式文件系统

Hadoop Distributed File Systems (HDFS)

HDFS就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件

HDFS用来存储数据

- HDFS is a file system that
  - 遵循主从架构
  - follows master-slave architecture
  - 允许我们在多个节点(机器)上存储数据
  - allows us to store data over multiple nodes (machines) ,
  - 允许多个用户访问数据
  - allows multiple users to access data.
  - just like file systems in your PC

- HDFS supports
  - distributed storage 分布式存储
  - distributed computation 分布式计算
  - horizontal scalability 水平可伸缩性；水平扩展性

垂直扩展 与 水平扩展

Vertical Scaling vs. Horizontal Scaling

Vertical Scaling

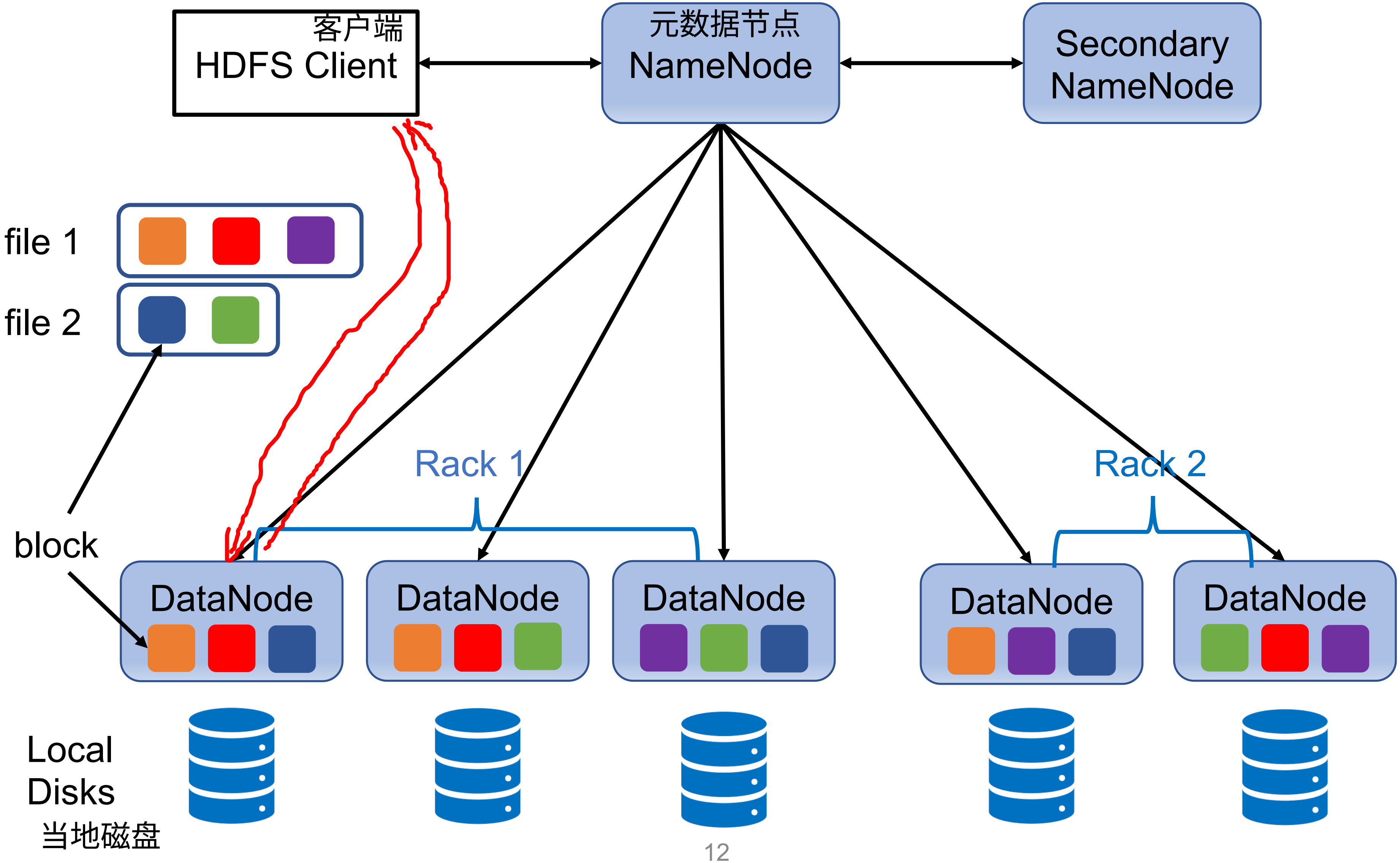
垂直扩展

Horizontal Scaling

水平扩展

体系结构

HDFS Architecture



NameNode 是一个通常在 HDFS 实例中的单独机器上运行的软件。它负责管理文件系统名称空间和控制外部客户机的访问。NameNode 决定是否将文件映射到 DataNode 上的复制块上。

NameNode

不存储数据

NameNode维护和管理DataNodes（从节点）中的块

- NameNode maintains and manages the blocks in the DataNodes (slave nodes).
  - Master node

- Functions:
  - 记录所有文件的元数据
  - records the metadata of all the files
    - FsImage: file system namespace

      NameNode 在一个称为 FsImage 的文件中存储所有关于文件系统名称空间的信息
    - EditLogs: all the recent modifications
  - records each change to the metadata
  - regularly checks the status of datanodes
  - keeps a record of all the blocks in HDFS
  - if the DataNode failure, handle data recoveray

    如果DataNode发生故障，请处理数据恢复

包含所有事务的记录文件，FsImage 和 EditLog 文件也需要复制副本，以防文件损坏或 NameNode 系统丢失

DataNode

商品硬件存储数据

- A commodity hardware stores the data
  - Slave node

从节点；从属节点

DataNode 也是一个通常在 HDFS实例中的单独机器上运行的软件。Hadoop 集群包含一个 NameNode 和大量 DataNode。DataNode 通常以机架的形式组织，机架通过一个交换机将所有系统连接起来。Hadoop 的一个假设是：机架内部节点之间的传输速度快于机架间节点的传输速度 [5] 。DataNode 响应来自 HDFS 客户机的读写请求。它们还响应来自 NameNode 的创建、删除和复制块的命令。NameNode 依赖来自每个 DataNode 的定期心跳（heartbeat）消息。每条消息都包含一个块报告，NameNode 可以根据这个报告验证块映射和其他文件系统元数据。如果 DataNode 不能发送心跳消息，NameNode 将采取修复措施，重新复制在该节点上丢失的块

- Functions
  - stores actual data 存储实际数据
  - perform the read and write requests 执行读写请求
  - report the health to NameNode (heartbeat)

NameNode vs. DataNode

	NameNode	DataNode
Quantity	One	Multiple
Role	Master	Slave 奴隶
Stores	Metadata of files	Blocks
硬件要求 Hardware Requirements	大容量内存 High Capacity Memory	高容量硬盘 High Volume Hard Drive
Failure rate	Lower	Higher
Solution to Failure	Secondary NameNode	Replications 复制；拷贝

If NameNode failed…

- All the files on HDFS will be lost
  - there's no way to reconstruct the files from the blocks in DataNodes without the metadata in NameNode

为了使NameNode能够抵抗故障

- In order to make NameNode resilient to failure

备份在主节点上的元数据

  - back up metadata in NameNode (with a remote NFS mount)
  - Secondary NameNode

Secondary NameNode

取得存在于NameNode上的文件系统元数据的检查点

- Take checkpoints of the file system metadata present on NameNode

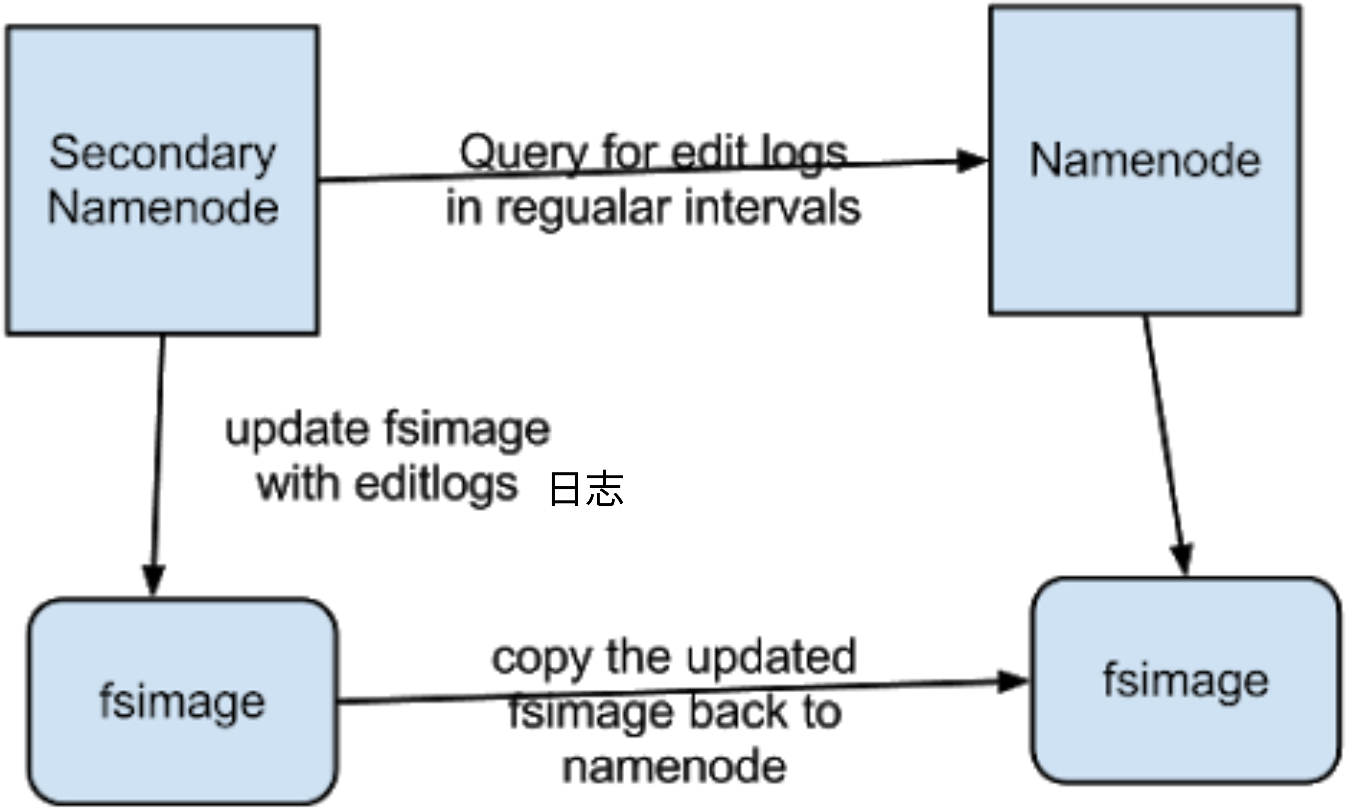
  它不是备份的NameNode

  - It is not a backup NameNode!

- Functions:
  - Stores a copy of FsImage file and Editlogs

    定期将编辑日志应用于FsImage并刷新编辑日志
  - Periodically applies Editlogs to FsImage and refreshes the Editlogs. 编辑日志
  - If NameNode is failed, File System metadata can be recovered from the last saved FsImage on the Secondary NameNode.

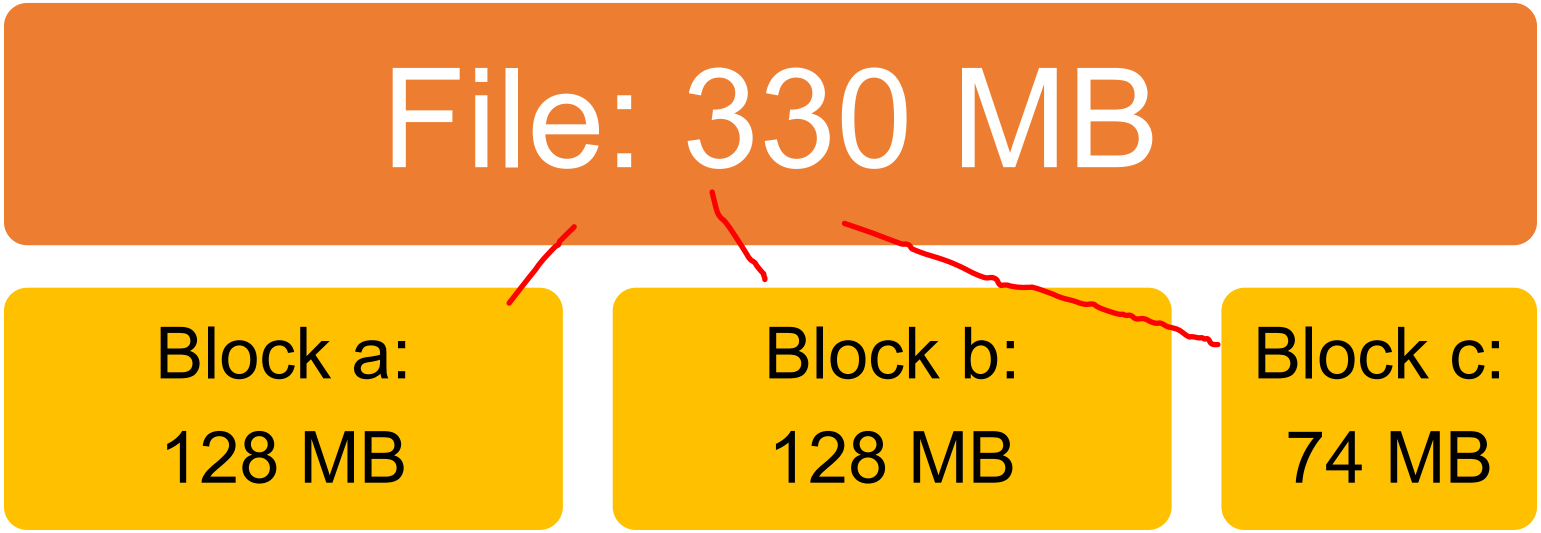
NameNode vs. Secondary NameNode


Secondary Namenode
Query for edit logs in regualar intervals
Namenode
update fsimage with editlogs
日志
fsimage
copy the updated fsimage back to namenode
fsimage

Blocks

块是存储数据的字节序列

- Block is a sequence of bytes that stores data
  - Data stores as a set of blocks in HDFS
  - Default block size is 128MB (Hadoop 2.x and 3.x)
  - A file is spitted into multiple blocks



File: 330 MB

Block a: 128 MB

Block b: 128 MB

Block c: 74 MB

Why Large Block Size?

- HDFS stores huge datasets
- If block size is small (e.g., 4KB in Linux), then the number of blocks is large:
  - too much metadata for NameNode
  - too many seeks affect the read speed
  - harm the performance of MapReduce too

    也会损害MapReduce的性能

- We don’t recommend using HDFS for small files due to similar reasons.
  - Even a 4KB file will occupy a whole block.

If DataNode Failed…

商品硬件故障

- Commodity hardware fails

  如果10分钟没有收到DataNode的NameNode消息，则认为该DataNode已删除

  - If NameNode hasn’t heard from a DataNode for 10mins, The DataNode is considered dead…

HDFS通过生成数据的多次复制来保证数据可靠性

- HDFS guarantees data reliability by generating multiple replications of data

  每个块默认有3个复制

  - each block has (3 replications by default)

  复制

  - replications will be stored on different DataNodes
  - if blocks were lost due to the failure of a DataNode, they can be recovered from other replications

    可以从其他复制中恢复它们

  - the total consumed space is 3 times the data size

    总消耗空间是数据大小的3倍

- It also helps to maintain data integrity

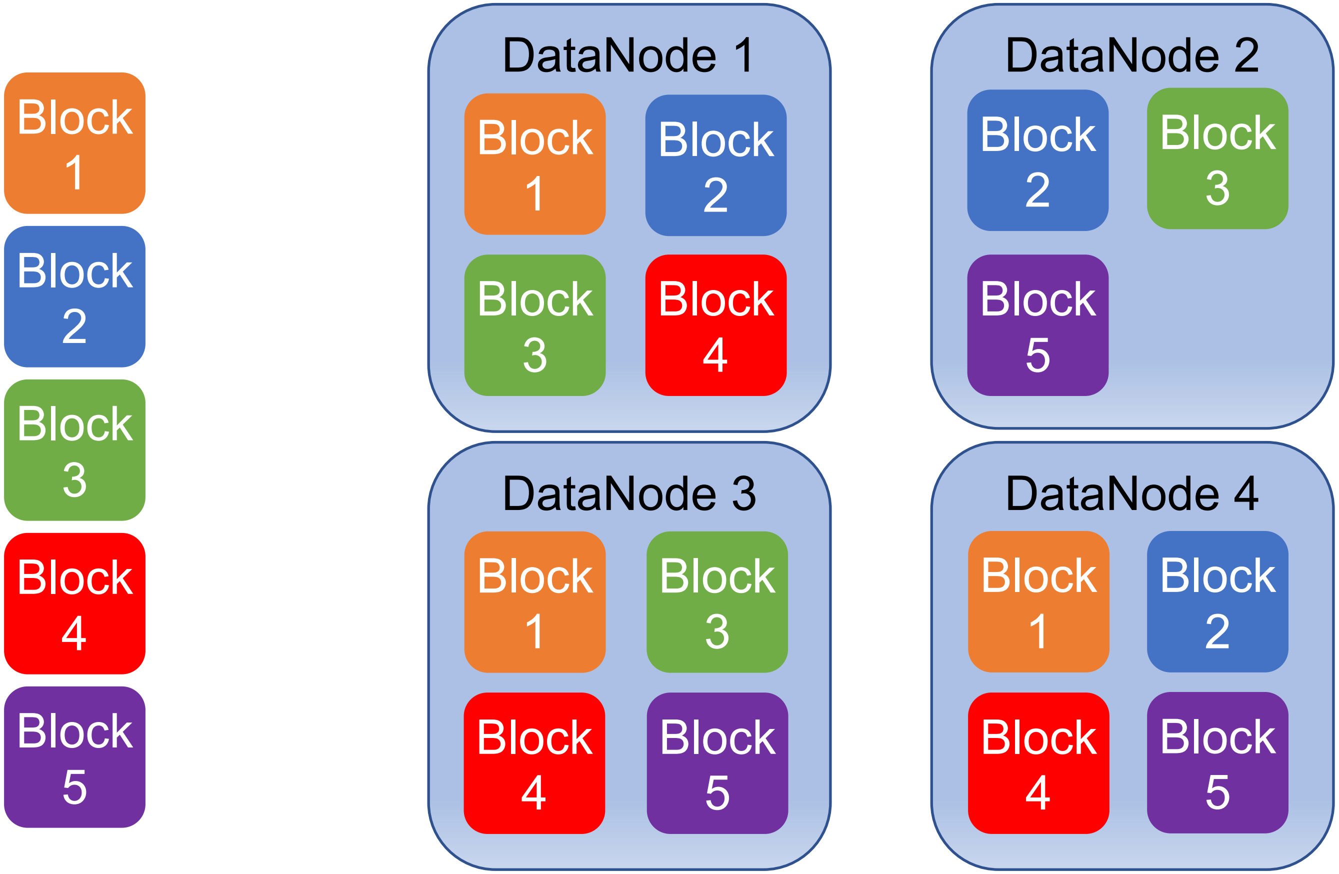
  它还有助于保持数据完整性

复制管理；复写管理

Replication Management

每个块被复制3次并存储在不同的DataNodes上

- Each block is replicated 3 times and stored on different DataNodes



为什么默认复制3次

Why default replication factor = 3?

- If 1 replicate
  - DataNode fails, block lost
- Assume
  - # of nodes N = 4000
  - # of blocks R = 1,000,000
  - Node failure rate FPD = 1 per day 每个节点上这么多

B=

- If one node fails, then R/N = 250 blocks are lost
  - E(# of losing blocks in one day) = 250
- Let the number of losing blocks follows Poisson distribution, then 让损失的块数遵循泊松分布
  - Pr[# of losing blocks in one day >= 250] = 0.508

Why default replication factor = 3?

- Assume
  - # of nodes N = 4000
  - Capacity of each node GB = 4000 Gigabytes
  - # of block replicas R = 1,000,000 * 3
  - Node failure rate FPD = 1 per day
  - Replication speed = 1.35 MB per second per node
- If one node fails, B = R/N = 750 replicas/blocks are unavailable
- There are on average S = 2B/(N-1) = 0.38 replicas per node for the blocks in the failed node

第一个失败节点中的块的每个节点副本

- So if second node fails, 0.38 blocks now have only a single replica

Why default replication factor = 3?

- If the third node fails,
  - The probability that it has the only remaining replica of a particular block is
    - Pr[last] = 1/(N-2) = 0.000250
  - The probability that it has none of those replicas is
    - Pr[none] = $(1\text{-Pr[last]})^S$ = 0.999906
  - The probability of losing the last replica of a block is
    - Pr[lose] = 1- Pr[none] = 9.3828E-05

- Recall:
  - N is # of nodes
  - S is the # of replicas per node for the blocks in the first failed node

Why default replication factor = 3?

- Assume # of node failures follows Poisson distribution with rate
  - ω=FPD/(24*3600)=1.1574E-05 per second

重新复制是对其余节点的完全并行操作

- Re-replication is a fully parallel operation on the remaining nodes
  - Recovery (re-create the lost replicas) time is
    - 1000 * GB / MPS / (N-1) = 740.93 seconds
    - Recovery rate μ= 1/ 740.93 per second
  - E(# of failed nodes in 1 sec) =ω/μ = 0.008576
- At any second, the probability of k failed nodes follows Poisson distribution
  - Pr[0 failed node] = 0.991461
  - Pr[1 failed node] = 0.008502
  - Pr[2 or more failed nodes] = 1- Pr(0) - Pr(1) = 0.00003656
- Thus, the rate of third failure is
  - Pr[2 or more failed nodes] *ω= 4.2315E-10 per sec
- The rate of losing a data block is
  - λ=Pr[2 or more failed nodes] *ω* Pr[lose] = 3.9703E-14

Why default replication factor = 3?

回想一下，一秒钟内丢失数据块的速率是

- Recall that in one second, the rate of losing a data block is
  - λ = 3.9703E-14 per second

根据指数分布

- According to exponential distribution, we have:
  - Pr[losing a block in one year] = $1-e^{-\lambda t}$ = 0.00000125
    - t = 365*24*3600

- So replication factor = 3 is good enough.

同时失败呢

What about Simultaneous Failure?

- If one node fails, we’ve lost B (first) replicas
- If two nodes fail, we’ve lost some second replicas and more first replicas
- If three nodes fail, we’ve lost some third 复制品 replicas, some second replicas and some first replicas
- …

同时失败

What about Simultaneous Failure?

同时地

- Assume k of N nodes have failed simultaneously, let there be
  - L1(k,N) blocks have lost one replica 复制品
  - L2(k,N) blocks have lost two replicas
  - L3(k,N) blocks have lost three replicas
  - B is # of unavailable blocks if one node fails
- k=0:
  - L1(0,N) = L2(0,N) = L3(0,N) = 0
- k=1:
  - L1(1,N) = B
  - L2(1,N) = L3(1,N) = 0
- k=2:
  - L1(2,N) = 2B-2*L2(2,N)
  - L2(2,N) = 2*L1(1,N)/(N-1)
  - L3(2,N) = 0
- k=3:
  - L1(3,N) = 3B-2*L2(3,N)-3*L3(3,N)
  - L2(3,N) = 2*L1(2,N)/(N-2)+L2(2,N)-L3(3,N)
  - L3(3,N) = L2(2,N)/(N-2)

$B= R/N$

What about Simultaneous Failure?

老师在7月9号更改了这一块的内容

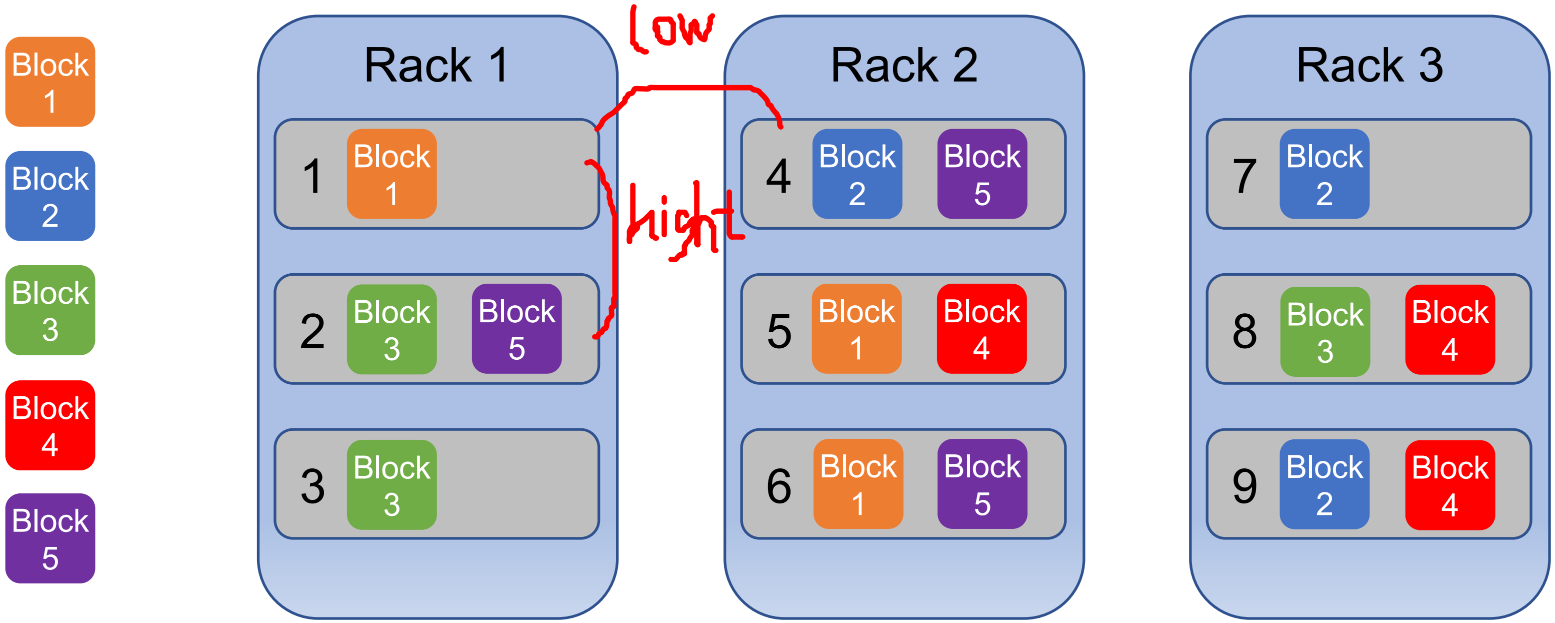
- In general
  - $L1(k,N) = k*B-2*L2(k,N)-3*L3(k,N)$
  - $L2(k,N) = 2*L1(k-1,N)/(N-k+1)+L2(k-1,N)-$ ~~$L3(k,N)$~~ $L2(k-1,N)/(N-k+1)$
  - $L3(k,N) = L2(k-1,N)/(N-k+1)+L3(k-1,N)$
- Let $N = 4000$, $B = 750$, we have

Failed Nodes	1st replicas lost	2nd replicas lost	3rd replicas lost
50	36,629	433	2
100	72,002	1,479	13
150	107,374	2,504	39
200	143,963	2,905	76

机架感知算法

Rack Awareness Algorithm

- If the replication factor is 3:
  - 1st replica will be stored on the local DataNode
  - 不同的机架
  - 2nd on a different rack from the first.
  - 3rd on the same rack as 2nd, but on a different node.

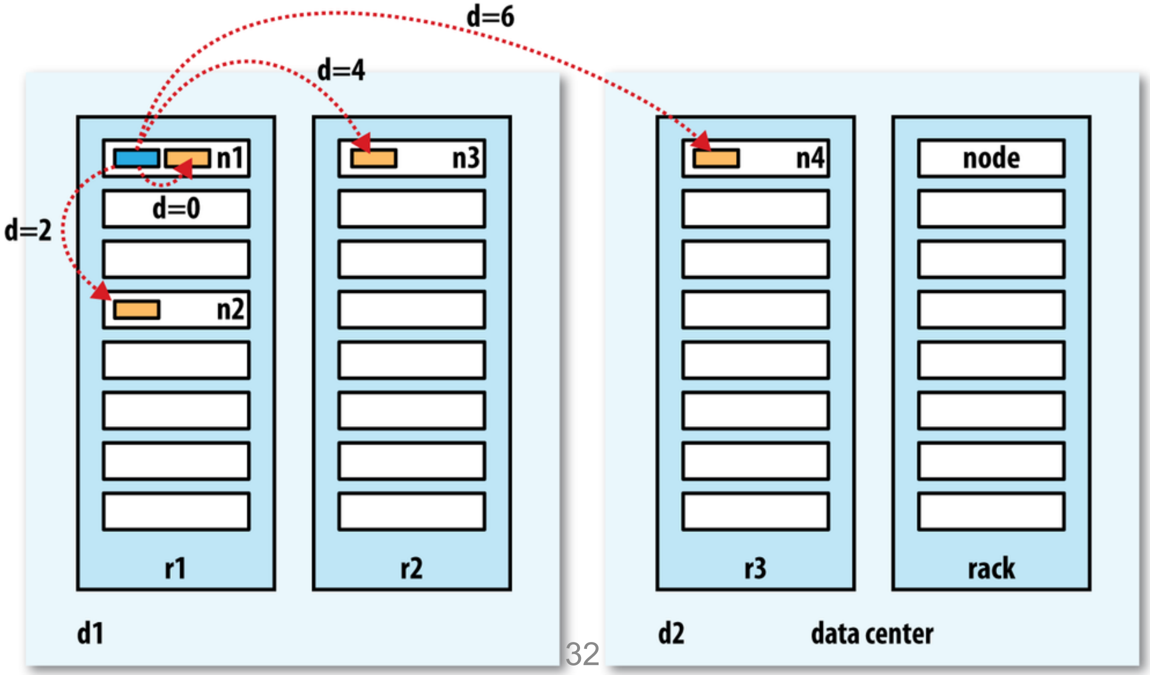


为什么要了解机架

Why Rack Awareness?

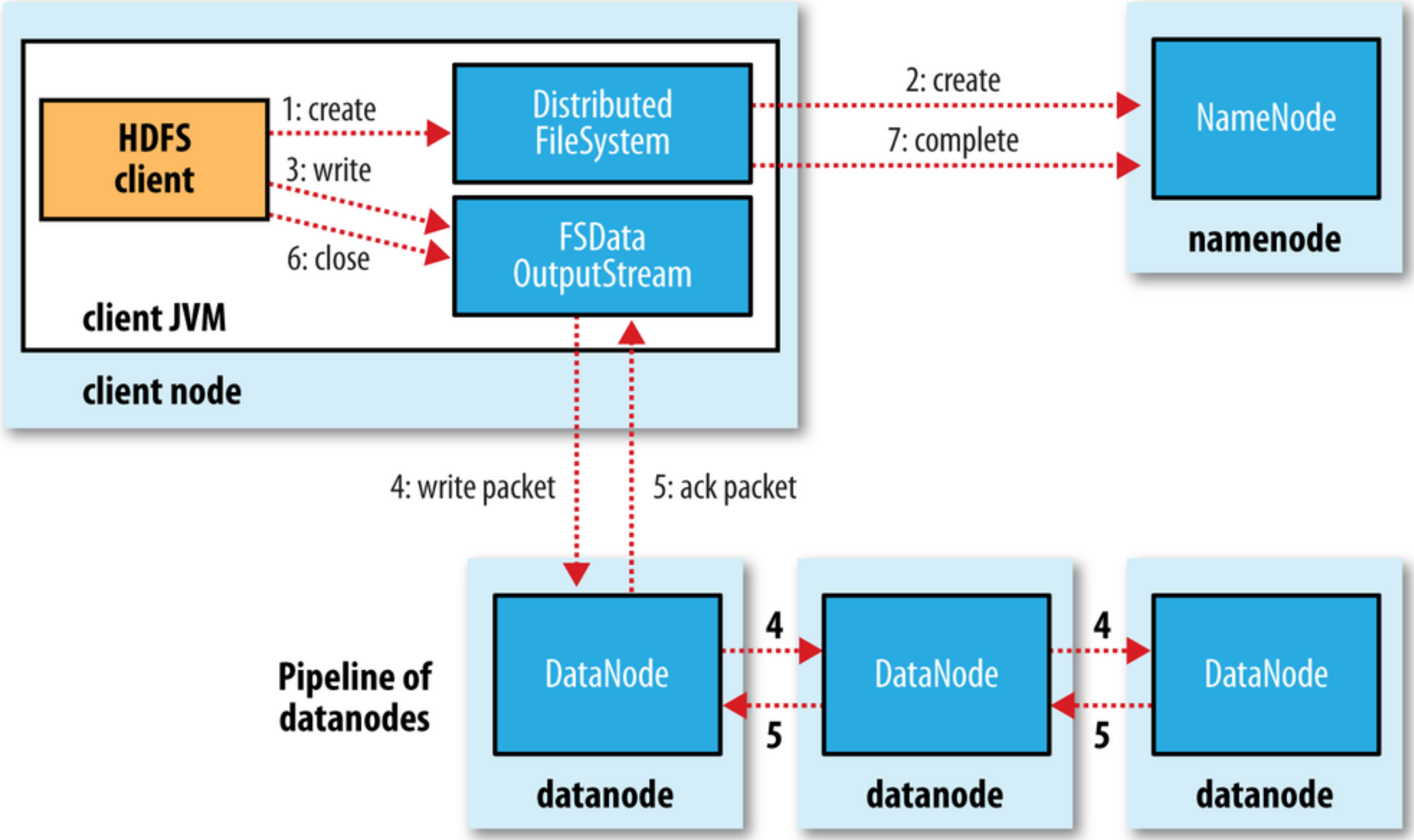
减少延迟

- Reduce latency
  - Write: to 2 racks instead of 3 per block
  - Read: blocks from multiple racks
- Fault tolerance 容错功能
  - Never put your eggs in the same basket



Write in HDFS

- Create file – Write file – Close file



Write in HDFS

任何时候都只允许单人作家

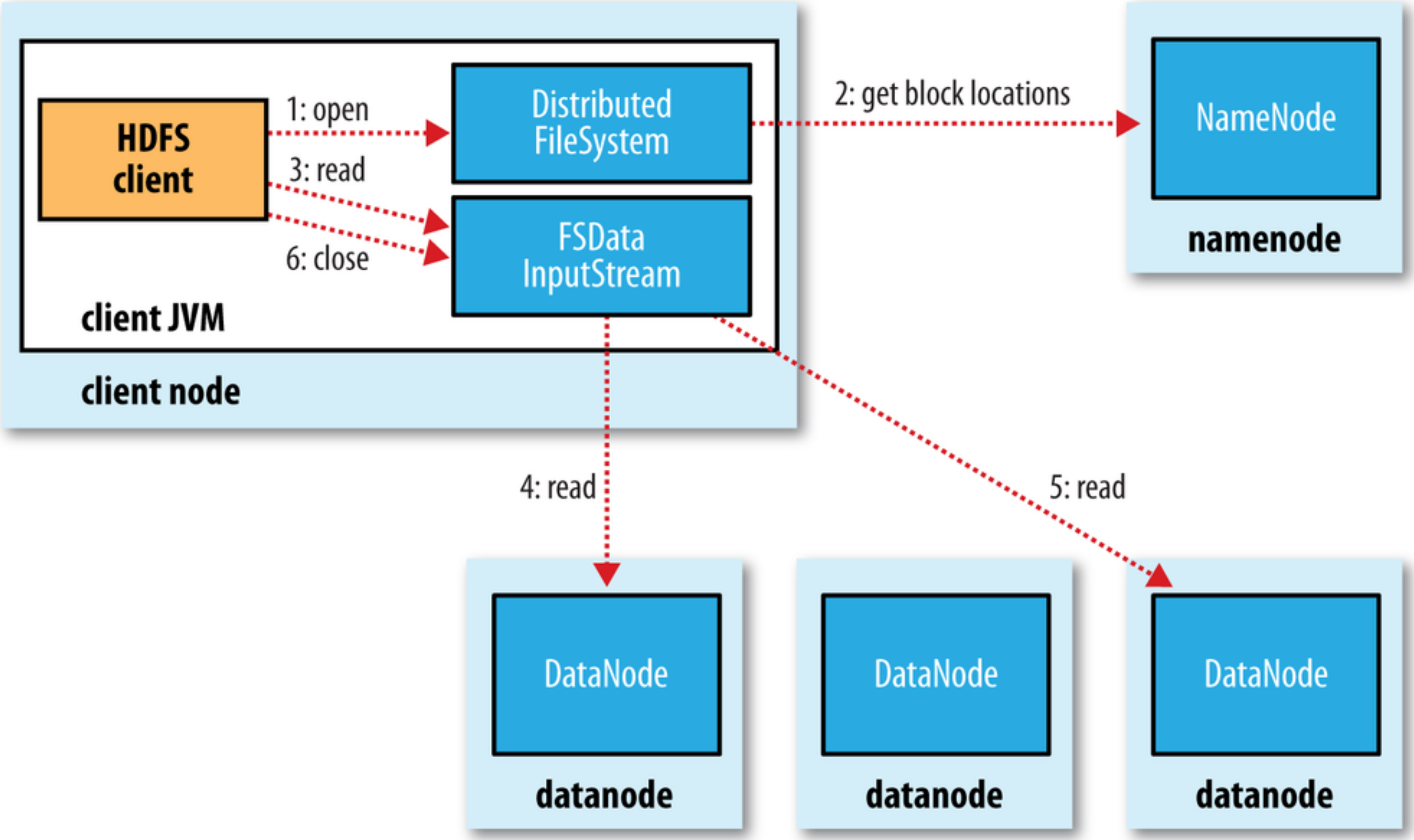
- There is only single writer allowed at any time

这些块是同时写入的

- The blocks are writing simultaneously
- For one block, the replications are replicating sequentially 对于一个块，复制是按顺序复制的
- The choose of DataNodes is random, based on replication management policy, rack awareness, …

DataNodesis的选择是随机的，基于复制管理策略，机架感知

Read in HDFS


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

Read in HDFS

允许多个读者同时阅读

- Multiple readers are allowed to read at the same time

块正在同时读取

- The blocks are reading simultaneously

始终选择与客户端最近的DataNode（基于网络拓扑）

- Always choose the closest DataNodes to the client (based on the network topology)

处理错误和损坏的块

- Handling errors and corrupted blocks

  避免再次访问dataNode

  - avoid visiting the dataNode again
  - report to NameNode