# Hadoop整体架构

大数据生态需要解决的问题：

1. 大数据框架之间的通信
2. 系统的容灾和高可用
3. 系统横向可扩展性
4. 高吞吐率/性能/数据分布/负载均衡

Hadoop体系下的三大组件:hdfs、yarn、mapReduce,其中mapReduce无需单独部署,Hadoop体系下有两个集群hdfs、yarn,其功能全部在hadoop包下,部署时只需改变其角色为hdfs或者yarn即可

# HDFS

## 1.hdfs特点

**HDFS 是基于流数据模式访问和处理超大文件的需求而开发的，它可以运行于廉价**

**的商用服务器上。HDFS 在设计时的假设和目标包括以下几个方面:**

**硬件出错**:Hadoop 假设硬件出错是一种正常的情况，而不是异常，为的就是在硬件出错的情况下尽量保证数据完整性，HDFS 设计的目标是在成百上千台服务器中 存储数据，并且可以快速检测出硬件错误和快速进行数据的自动恢复。

**流数据读写**:不同于普通的文件系统，Hadoop 是为了程序批量处理数据而设计的，而不是与用户的交互或者随机读写，所以 POSIX 对程序增加了许多硬性限制，程序必须使用流读取来提高数据吞吐率。

**大数据集**:HDFS 上面一个典型的文件一般是用GB或者TB计算的，而且一个数百

台机器组成的集群里面可以支持过千万这样的文件。

**简单的文件模型**:HDFS 上面的文件模型十分简单，就是一次写入多次读取的模型，文件一旦创建，写入并关闭了，之后就再也不会被改变了，只能被读取，这种模型刚好符合搜索引擎的需求，1.04后已有追加写入功能。

**强大的跨平台兼容性**:由于是基于 Java 的实现，无论是硬件平台或者是软件平台要求都不高，只要是 JDK 支持的平台都可以兼容。

**HDFS局限性，主要表现在以下几个方面:**

**不适合低延迟数据访问**:如果要处理一些用户要求时间比较短的低延迟应用请求，则 HDFS 不适合。HDFS 是为了处理大型数据集分析任务，主要是为了达到较高的数据吞吐量而设计的，这就可能以高延迟作为代价。目前的一些补充的方案，比如 使用 HBase，通过上层数据管理项目来尽可能地弥补这个不足。

**无法高效存储大量小文件**:在 Hadoop 中需要使用 NameNode(目录节点)来管理 文

件系统的元数据，以响应客户端请求返回文件位置等，因此，文件数量大小的限制要

由 NameNode 来决定。例如，每个文件、索引目录及块大约占 100 字节，如 果有 100

万个文件，每个文件占一个块，那么，至少要消耗 200MB 内存，这似乎 还可以接

受。但是，如果有更多文件，那么，NameNode 的工作压力更大，检索处理元数据的

时间就不可接受了。

**不支持多用户写入及任意修改文件**:在 HDFS 的一个文件中只有一个写入者，而且写操作只能在文件末尾完成，即只能执行追加操作。目前 HDFS 还不支持多个用户对同一文件的写操作，以及在文件任意位置进行修改。

## 2.hdfs架构

**Block**

HDFS 中的文件在物理上是分块存储（block），块的大小可以通过配置参数( dfs.blocksize)来规定，默认大小在 hadoop2.x 版本中是 128M，老版本中是 64M

**Namenode**

namenode 是 HDFS 集群主节点，负责维护整个 hdfs 文件系统的目录树，以及每一个路径（文件）所对应的 block 块信息（block 的 id，及所在的 datanode 服务器）

Hadoop2.0之后nameNode支持高可用,从节点只做主节点数据备份,不提供服务,edits的写入使用paxos算法实现

**Secondarynode**

是一个镜像，分担 namenode 的工作量；是 NameNode 的冷备份；合并 fsimage 和 fsedits 然后再发给 namenode。

**Fsimage**：元数据镜像文件（文件系统的目录树）

**edits**：元数据的操作日志（针对文件系统做的修改操作记录）

namenode 内存中存储的是=fsimage+edits

SecondaryNameNode 负责定时默认 1 小时，从 namenode 上，获取 fsimage 和

edits 来进行合并，然后再发送给 namenode。减少 namenode 的工作量。

**Datanode**

文件的各个 block 的存储管理由 datanode 节点承担，datanode 是 HDFS 集群从节点，每一个 block 都可以在多个 datanode 上存储多个副本（副本数量也可以通过参数设置 dfs.replication）。HDFS 是设计成适应一次写入，多次读出的场景，且不支持文件的修改。

### hdfs容灾

1、在目录节点和数据节点之间维持心跳检测。当由于网络故障之类的原因，

导致数 据节点(DataNode)发出的心跳包没有被目录节点(NameNode)正常收

到的时候，目录节点就不会将任何新的 IO 操作派发给那个数据节点，该

数据节点上的数据被认为是无效的， 因此，目录节点会检测是否有文件块

的副本数目小于设置值，如果小于就自动开始复制新的副本，并分发到其

他数据节点上。

2、检测文件块的完整性。HDFS 会记录每个新创建的文件的所有块的校验和。

当以后检索这些文件的时候，从某个节点获取块，会首先确认校验和是否

一致，如果不一致，会从其他数据节点上获取该块的副本。

3、集群的负载均衡。由于节点的失效或者增加，可能导致数据分布的不均

匀，当某个数据节点的空闲空间大于一个临界值的时候，HDFS 会自动从

其他数据节点迁移数据过来。

4、维护多个 FsImage 和 Editlog 的拷贝。目录节点上的 fsimage 和 edits 日志文

件是 HDFS 的核心数据结构，如果这些文件损坏了，HDFS 将失效。因而，目录节点可以配置成

## hdfs常用api

常用的api有两种WebHDFS,HttpFS

WebHDFS 是 HortonWorks 开发的，然后捐给了 Apache，目前是 Hdfs 内置的服务，

默认处于开启状态。当 client 请求某文件时，WebHDFS 会将其重定向到该资源所在的datanode。

HttpFS 是 HDFS 一个独立的服务，HttpFS 是 Cloudera 开发的，也捐给了 Apache。

两者都是 HDFS 的 REST API，但稍有差异：当 client 请求某文件时，WebHDFS 会将其重

定向到该资源所在的 datanode，而 HttpFs 相等于一个“网关”，所有的数据先传输到

该 httpfs server，再由该 httpfs server 传输到 client,此种情况多了数据传输的次数,如果数据量比较大严重影响性能,并消耗带宽

# Yarn

**ResourceManager**：ResourceManager 是一个中心的服务，它做的事情是调度、启动每一个Job所属的ApplicationMaster，并且监控ApplicationMaster的存在情况。ResourceManager 负责作业与资源的调度，接收 JobSubmitter 提交的作业，按照作业的上下文(Context) 信息以及从NodeManager 收集来的状态信息，启动调度过程，分配一个Container作为ApplicationMaster。

ResourceManager使用zk来维持集群高可用,使用zk选主,配合zk与acl解决脑裂问题

**NodeManager**：功能比较专一，就是负责Container状态的维护，并向ResourceManager 保持心跳,具体的执行作业服务器,具体为创建task的容器,并执行map或者reduce操作。

**ApplicationMaster**：负责一个 Job 生命周期内的所有工作，类似老的框架中JobTracker。但是要注意，每一个Job(不是每一种)都有一个 ApplicationMaster，它可以运行在 ResourceManager 以外的机器上。

**Yarnchild**：MrAppmaster 运行程序时向 resouce manager 请求的maptask/reduceTask；

**Yarn的执行流程**

1、客户端发出请求，YARNRUNNER 接受，生成一个代理对象，向 resource manager请求一个 application

2、resource manager 返回 application 的提交路径和application\_id(这里使用id是

因为可能有多个任务用id来区别)

3、YARNRUNNER 向 hdfs 提交job运行所需要的文件(application，

job.split,job,.xml,job.jar)

4、向 resource manager 报告提交完成，申请一个 mrAppMaster

5、将用户的请求初始化成一个task，将 task 放到队列中，等待 node manager 来

领取 task 任务。(这其中使用了调度策略，节约资源，如：Fair Capacity 等等)

6、node manager 领取到任务，生成一个 Container，然后在 hdfs 中下载运行资源。

7、向 resource manager 申请运行 maptask 的容器

8、其他的 node manager 领取到 resouce manager 的任务，创建容器，此时的Container 则是 YarnChild,也是 maptask，然后 maptask 在 hdfs 下载所要运行的资源。

9、MrAppMaster 发送程序脚本运行 jar，当 maptask 中的程序运行完成后，maptask 的资源被 resource manager 回收了，但跑完的资源在 node manager中。

10、当 maptask 运行完成后 MRAppmaster 又向 resorce manager 申请 reduce task

（至于它申请多少个是由它有多少个 map task 决定的），然后根据忙于不忙 node manager 领取任务。

11、redcuetask 向 map 获取相应分区的数据资源，运行文件。

12、application 运行完毕后 MrAppmaster 会向 resource manager

# mapReduce

### 分治算法

分治法的设计思想是：

分–将问题分解为规模更小的子问题；

治–将这些规模更小的子问题逐个击破；

合–将已解决的子问题合并，最终得出“母”问题的解；

一个先自顶向下，再自底向上的过程。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。

分治算法所能解决的问题一般具有以下几个特征：

1) 该问题的规模缩小到一定的程度就可以容易地解决

2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性

质。

3) 利用该问题分解出的子问题的解可以合并为该问题的解；

4) 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子

问题

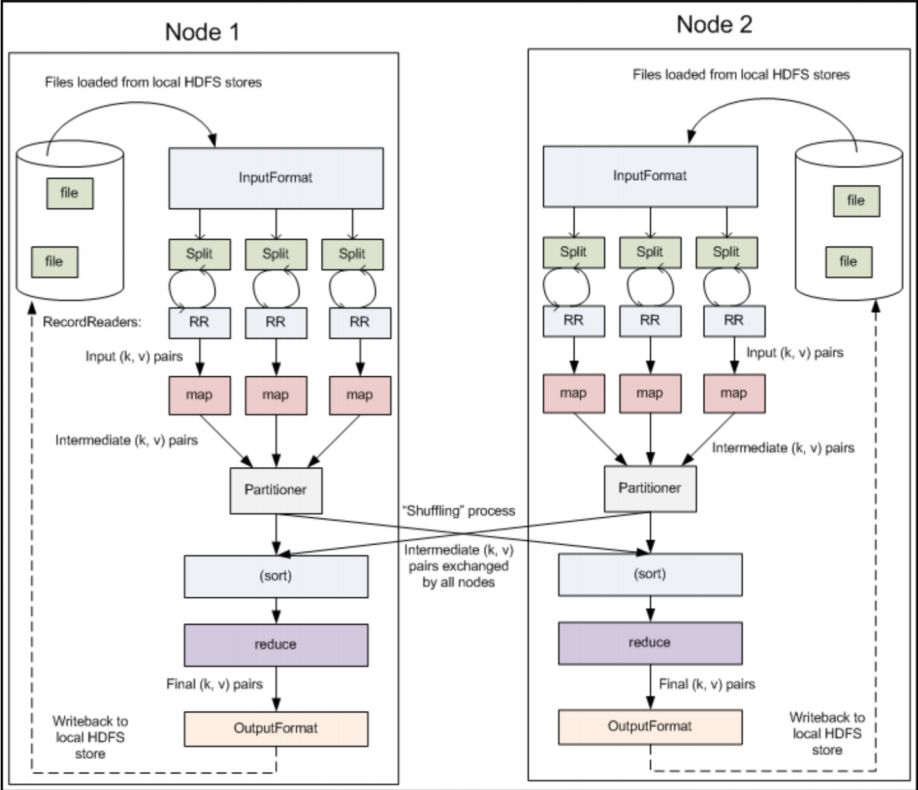
### 分布式计算最大优点:

可以很容易地通过增加计算机来扩充新的计算节点，并由此获得不可思议的海量计算能力，同时又具有相当强的容错能力，一批计算节点失效也不会影响计算的正常进行以及结果的正确性。Google 就是这么做的，他们使用了叫做 MapReduce 的并行编程模型进行分布式并行编程，运行在叫做 GFS ( Google File System )的分布式文件系统上， 为全球亿万用户提供搜索服务。

MapReduce 是 Google 公司的核心计算模型，这是一个令人惊讶的、简单却又威力巨大的模型，它将复杂的、运行于大规模集群上的并行计算过程高度地抽象到了两个函数:Map和Reduce。

MapReduce 计算模型的核心是 map 和 reduce 两个函数，这两个函数由用户负责 实现， 功能是按一定的映射规则将输入的<key，value>转换成另一个或一批<key，value>输出

### mapReduce流程:



• 输入文件

文件是 MapReduce 任务的数据的初始存储地。正常情况下，输入文件一般是存在HDFS里。这些文件的格式可以是任意的;我们可以使用基于行的日志文件，也可以使用二进制格式，多行输入记录或其它一些格式。这些文件会很大(数十GB或更大)。

• 输入格式

InputFormat 类定义了如何分割和读取输入文件，它提供了以下几个功能:

(1) 选择作为输入的文件或对象;

(2) 定义把文件划分到任务的 InputSplits;

(3) 为 RecordReader 读取文件提供了一个工厂方法。

• 输入块（InputSplit）

一个输入块描述了构成 MapReduce 程序中单个 map 任务的一个单元。把一个 MapReduce 程序应用到一个数据集上，即是指一个作业，会由几个(也可能几百个)任务组成。Map任务可能会读取整个文件，但一般是读取文件的一部分。默认情况下，FileInputFormat及其子类会以128MB为基数来拆分文件(与 HDFS的Block默认大小相同，Hadoop建议Split大小与此相同)。

• 记录读取器（RecordReader）

InputSplit 定义了如何切分工作，但是没有描述如何去访问它。RecordReader 类则是实际地用来加载数据，并把数据转换为适合mapper读取的键值对。RecordReader实例是由输入格式定义的，默认的输入格式 TextInputFormat，提供了一个LineRecordReader，这个类会把输入文件的每一行作为一个新的值，关联到每一行的键则是该行在文件中的字节偏移量。 RecordReader 会在输入块上被重复地调用，直到整个输入块被处理完毕，每一次调用 RecordReader 都会调用 Mapper 的 map()方法。

• Mapper

Mapper 执行了 MapReduce 程序第一阶段中有趣的用户定义的工作。给定一个键值对，map()方法会生成一个或多个键值对，这些键值对会被送到 Reducer 那里。对于整个作业输入部分的每一个map任务(输入块)，每一个新的 Mapper 实例都会在单独的Java进程中被初始化，mapper 之间不能进行通信。这就使得每一个map任务的可靠性不受其它map任务的影响，只由本地机器的可靠性来决定。

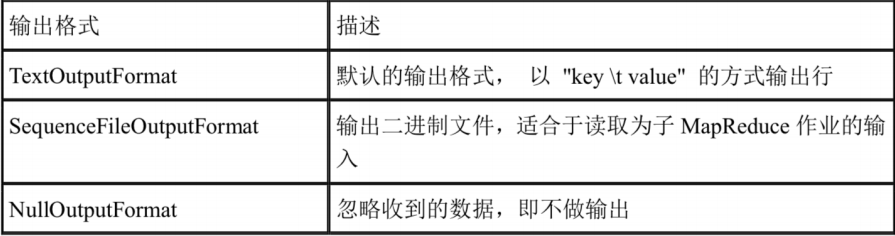
• Partition&Shuffle

当第一个 map 任务完成后，节点可能还要继续执行更多的 map 任务，但这时候也开始把 map任务的中间输出交换到需要它们的reducer那里去，这个把“map 输出” 移动到 reducer 那里去的过程就叫做 shuffle。每一个reduce节点会被分配得到“map 输出的键集合”中的一个不同的子集合，这些子集合(被称为“partitions”)是 reduce任务的输入数据。每一个map任务生成的键值对，可能会隶属于任意的partition，有着相同键的数值总是在一起被 reduce，不管它是来自那个 mapper 的。因此，所有的 map节点必须就把不同的中间数据发往何处达成一致。Partitioner 类就是用来决定给定键值对的去向，默认的分类器(partitioner) 会计算键的哈希值，并基于这个结果来把键赋到相应的 partition上。

• Reducer

每一个reduce任务负责对那些关联到相同键上的所有数值进行归约(reducing)，每一 个节点收到的中间键集合，在被送到具体的 reducer 那里前就已经自动被 Hadoop排序过了。 每个 reduce 任务都会创建一个 Reducer 实例，这是一个用户自定义代码的实例，负责执行特定作业的第二个重要的阶段。对于每一个已被赋予到 reducer 的 partition 内的键来说，reducer 的 reduce()方法只会调用一次，它会接收一个键和关联到键的所有值的一个迭代器，迭代器会以一个未定义的顺序返回关联到同一个键的值。reducer 也要接收一个 OutputCollector 和 Report 对象，它们像在 map()方法中那样被使用。

• 输出格式



• RecordWriter

这个跟InputFormat中通过 RecordReader 读取单个记录的实现很相似，OutputFormat类 是RecordWriter对象的工厂方法，用来把单个的记录写到文件中，就像是OuputFormat直接写入的一样。

Reducer输出的文件会留在HDFS上供你的其它应用使用，比如，另外一个MapReduce作业，或者一个给人工检查的单独程序。

### Shuffle 过程详解

在 MapReduce 流程中，为了让reduce可以并行处理 map 结果，必须对 map 的输出进行一定的排序和分割，然后再交给对应的 reduce，而这个将map输出进行进一步整理并交给 reduce的过程，就称为shuffle。Shuffle过程是 MapReduce工作流程的核心，也被称为奇迹发生的地方。要想理解 MapReduce，Shuffle是必须要了解的。

shuffle 过程包含在map和reduce两端中，描述着数据从map task输出到reduce task的这段过程。在map端的shuffle过程是对map的结果进行划分(partition)、排序(sort)和spill(溢写)，然后，将属于同一个划分的输出合并在一起，并写到磁盘上，同时按照不同的划分将结果发送给对应的reduce。reduce端又会将各个map送来的属于同一个划分的输出进行合并(merge)，然后对合并的结果进行排序，最后交给reduce处理。

# Hbase

**列数据库与关系型数据库异同:**

1、数据类型:HBase只有简单的字符串类型，所有类型都是交由用户自己处理，它只保存字符串。而关系数据库有丰富的类型选择和存储方式。

2、数据操作:HBase 操作只有很简单的插入、查询、删除、清空等，表和表之间是分离的，没有复杂的表和表之间的关系，所以，不能也没有必要实现表和表之间的关联等操作。而传统的关系数据通常有各种各样的函数、连接操作。

3、存储模式:HBase 是基于列存储的，每个列族都有几个文件保存，不同列族的文件是分离的。传统的关系数据库是基于表格结构和行模式保存的。

4、数据维护:HBase 的更新，确切地说，应该不叫更新，而是一个主键或者列对应的新的版本，而它旧有的版本仍然会保留，所以，它实际上是插入了新的数据，而不是传统关系数据库里面的替换修改。

5、可伸缩性:HBase 和 BigTable 这类分布式数据库就是直接为了这个目的开发出来的，能够轻易地增加或者减少(在硬件错误的时候)硬件数量，而且对错误的兼容性较高。而传统的关系数据库通常需要增加中间层才能实现类似的功能。

**Hbase接口:**

1、Native Java API:最常规和高效的访问方式，适合 Hadoop MapReduce 作业并行批处理 HBase 表数据;

2、HBase Shell:HBase 的命令行工具，最简单的接口，适合 HBase 管理使用;

3、Thrift Gateway:利用 Thrift 序列化技术，支持 C++，PHP，Python 等多种语言，适合其他异构系统在线访问 HBase 表数据;

4、RESTGateway:支持 REST 风格的 HttpAPI 访问 HBase，解除了语言限制;

5、Pig:可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据，和 Hive 类

似，最终也是编译成 MapReduce 作业来处理 HBase 表数据，适合做数据统

计;

1. Hive:可以使用类似 SQL 语言来访问 HBase。

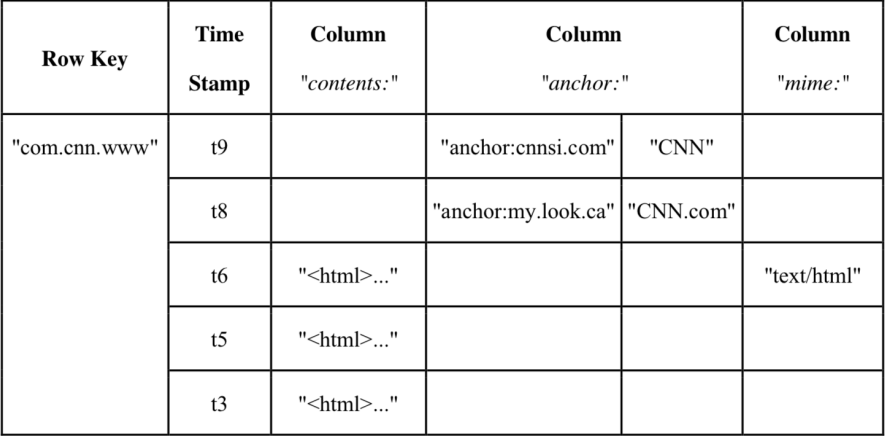
**HBase数据模型中，包括如下三个重要概念:**

行键(Row Key):HBase 表的主键，表中的记录按照行键排序;

时间戳(Timestamp):每次数据操作对应的时间戳，可以看作是数据的版本号;

列族(Column Family):表在水平方向有一个或者多个列族组成，一个列族中可以由任意多个列组成，即列族支持动态扩展，无需预先定义列的数量以及类型，列族相当于将MySQL的多张表整合到一张表里

所有列均以二进制格式存储，用户需要自行进行类型转换。





Client 与 HMaster 进行 RPC;对于数据读写类操作，Client 与 HRegionServer进行 RPC。

Zookeeper 中除了存储-ROOT-表的地址和 HMaster 的地址，HRegionServer也会把自己以“Ephemeral”方式注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个HRegionServer 的健康状态。

HMaster 在功能上主要负责表和 HRegion 的管理工作。

HRegionServer 主要负责响应用户I/O请求，向HDFS文件系统中读写数据，是HBase中最核心的模块。

HStore存储是HBase存储的核心了，由两部分组成，一部分是HMemStore，一部分是 HStoreFile。

任何时刻，一个 HRegion 只能分配给一个 HRegionServer。HMaster 记录了当前有哪些可用的 HRegionServer，以及当前哪些 HRegion 分配给了哪些HRegionServer，哪些 HRegion 还没有分配。

类似Mysql中的binlog，用来做灾难恢复，HLog记录数据的所有变更，一旦数据修改，就可以从HLog中进行恢复

473453787791 NZgrCb