RDMS与MapReduce区别？

为什么不能使用数据库对大量磁盘上的大规模数据进行批量分析呢？

因为，多年来磁盘存储容量快速增加的同时，其访问速度—磁盘数据读取速度—却未能与时俱进，即寻址时间的提高远远慢于传输速率的提高。寻址是将磁头移动到特定磁盘位置进行读写操作的过程。它是导致磁盘操作延迟的主要原因，而传输速率取决于磁盘的带宽。一个简单的减少读取时间的办法是同时从多个磁盘上读取数据

RDBMS比较适合于“点查询”和更新，数据集被索引后，数据库系统能够提供低延迟的数据检索和快速的少量数据更新。关系型数据库更适合持续更新的数据集

MapReduce比较适合以批处理的方式处理需要分析整个数据集的问题。MapReduce适合一次写入、多次读取数据的应用

MapReduce：批量查询处理器

MapReduce的核心假设之一：它可以进行高速的流式读写操作

MapReduce会尽量在计算节点上存储数据，以实现数据的本地快速访问。

MapReduce则在更高层次执行任务，即程序员仅从键值对函数的角度考虑任务的执行，这样数据流是隐含的。MPI赋予了程序员极大的控制能力，但是需要程序员显式控制数据流机制

MapReduce让程序员无需考虑系统的部分失效问题，因为自身的系统实现能够检测到失败的map或reduce任务，并让正常运行的机器重新执行这些失败的任务。前提条件：无共享框架，各个任务之间彼此独立。

MapReduce的设计目标是服务于那些只需数分钟或数小时即可完成的作业，并且运行于内部通过高速网络连接的单一数据中心内，并且该数据中心内的计算机需要由可靠的、定制的硬件构成

MapReduce任务过程被分为两个处理阶段：map阶段和reduce阶段。每个阶段都以键值对作为输入和输出，并由程序员选择它们的类型。程序员还需具体定义两个函数：map函数和reduce函数

Hadoop自身提供一套可优化网络序列化传输的基本类型，而不直接使用Java内嵌的类型

函数例子见page20

Java API控制的map函数一次只能处理一条记录，针对输入数据中的每一条记录，该框架均需调用Mapper的map()方法来处理

**Hadoop集群上运行作业时，需要将代码打包成一个JAR文件，Hadoop会在集群上分发这个文件。输入可以是单个文件，目录（此时将目录下所有文件当作输入）或符合特定文件模式的一组文件；输出是每个reducer输出一个文件。**

Hadoop的Streaming使用Unix标准流作为Hadoop和应用程序之间的接口

，而Streaming天生适合用于文本处理，在文本模式下使用时，它有一个数据的行视图。Map的输入数据通过标准输入流传递给map函数，并且是一行一行地传输，最后将结果行写到标准输出。Map输出的键值对是以一个制表符分隔的行。Reduce函数从标准输入流中读取输入行，该输入已由Hadoop框架根据键排过序，最后将结果写入标准输出。

HDFS：Hadoop分布式文件系统

当数据集的大小超过一台独立物理计算机的存储能力时，就有必要对它进行分区并存储到若干台单独的计算机上。管理网络中跨多台计算机存储的文件系统称为分布式文件系统。

HDFS以流式数据访问模式来存储超大文件，运行于商用硬件集群上

HDFS的构建思路：一次写入、多次读取是最高效的访问模式。数据集通常由数据源生成或从数据源复制而来，接着长时间在此数据集上进行各类分析。每次分析都将涉及该数据集的大部分数据甚至全部，因此读取整个数据集的时间延迟比读取第一条记录的时间延迟更重要。

HDFS不合适于：低时间延迟的数据访问，大量的小文件，多用户写入，任意修改文件

文件系统 = 文件系统块大小

= 磁盘块512字节 + 磁盘块512字节 + …

文件系统块大小

…

HDFS = 块64M + 块64M + … （磁盘----网络1）

块64M + 块64M + … （磁盘----网络2）

为何HDFS的块如此之大？

HDFS的块比磁盘块大，其目的是为了最小化寻址开销。如果块设置得足够大，从磁盘传输数据的时间可以明显大于定位这个块开始位置所需的时间。这样，传输一个由多个块组成的文件的时间取决于磁盘传输速率。MapReduce中的map任务通常一次处理一个块中的数据。

HDFS集群有两类节点：一个namenode（管理者） + 多个datanode（工作者）

namenode管理文件系统的命名空间。它维护着文件系统树及整棵树所有的文件和目录

datanode是文件系统的工作节点，它们根据需要存储并检索数据块（受客户端或namenode调度），并且定期向namenode发送它们所存储的块的列表

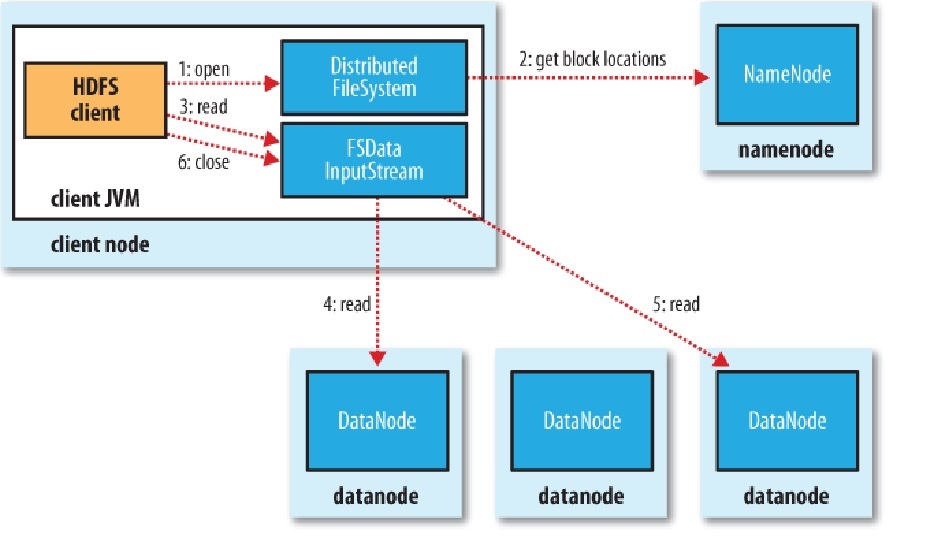
Hadoop对文件系统提供了许多接口，它一般使用URI方案来选取合适的文件系统实例进行交互。通过命令行接口或API接口操作HDFS

命令行接口：详细见手册

$hadoop fs –put local文件路径 分布式文件系统路径 //上传文件

客户端只需要读取连续的流，并且对于客户端都是透明的，以下是分析内部工作机制

文件读取剖析



简单地说：namenode告知客户端每个块中最佳的datanode,并让客户端直接联系该dataode且检索数据。

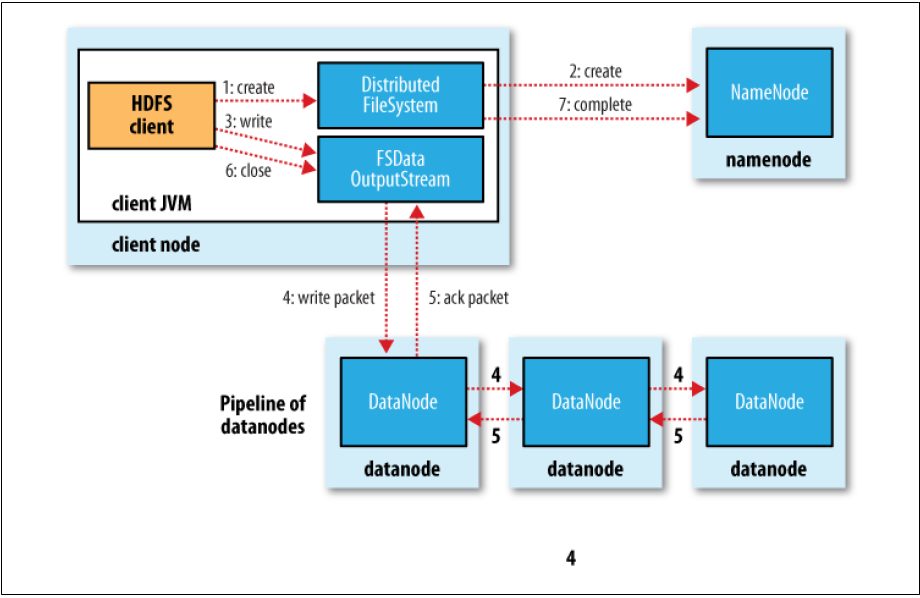
详细地说：

1）客户端通过调用FileSystem对象的open()方法来打开读取的文件，获取得HDFS的实例。HDFS通过RPC来调用namenode,以确定文件起始块的位置。返回一个FSDataInputStream对象给客户端

2）客户端调用FSDataInputStream对象的read()方法。通过存储着文件起始块的datanode地址连接距离最近的datanode.通过对数据流反复调用read()方法，可以将数据从datanode传输到客户端。到达块的末端时，会关闭与该datanode的连接，然后寻找下一个块的最佳datanode.

若在读取数据的时候，与datanode通信时遇到错误，它便会尝试从这个块的另外一个最邻近datanode读取数据。它也会记住那个故障datanode,以保证以后不会反复读取该节点上后续的块。

文件写入剖析



1）客户端通过对DistributedFileSystem对象调用create()函数来创建文件，DistributedFileSystem对namenode创建一个RPC调用，在文件系统的命名空间中创建一个新文件

2）namenode执行各种不同的检查以确保该文件不存在，且客户端有创建该文件的权限。DistributedFileSystem向客户端返回一个FSDataOutputStream对象

3）客户端写入数据时，FSDataOutputStream将它分成一个个的数据包，并写入内部队列，称为“数据队列”.根据datanote列表要求namenode分配适合的新块来存储数据备份。然后将数据包流式传输到管线中第1个datanode,该datanode存储数据包并将它发送到管线中的第2个datanode,…

4）客户端完成数据的写入后，会对数据流调用close()方法。该操作将剩余的所有数据包写入datanode管线中，并在联系namenode且发送文件写入完成信号之前，等待确认。Namenode已经知道文件由哪些块组成，所以它在返回成功前只需要等待数据块进行最小量的复制。

一致模型

文件系统的一致模型描述了对文件读写的数据可见性，写入文件的内容并不保证能立即可见，即使数据流已经刷新并存储。只有当写入的数据超过一个块后，新的reader才能看见第一个块，总之reader无视当前正在写入的块

HDFS提供sync()方法来强制所有的缓存与数据节点同步，当sync()方法返回成功后，对所有新的reader而言，HDFS能保证文件中到目前写入的数据均可见

每个文件均按块方式存储，每个块的元数据存储在namenode的内存中，因此Hadoop存储小文件会非常低效。因为大量的小文件会耗尽namenode中的大部分内存

Hadoop I/O

数据完整性

检测数据是否损坏的常见措施是：在数据第一次引入系统时计算校验和(checksum),并在数据通过一个不可靠的通道进行传输时再次计算检验和，这样就能发现数据是否损坏。注意检验和也是可能损坏的，不只是数据，但于由于校验和比数据小得多，所以损坏的可能性非常小。

常见的错误检测码是CRC-32（循环冗余检验），任何大小的数据输入均计算得到一个32位的整数校验和。

HDFS会对写入的所有数据计算校验和，并在读取数据时验证校验和。

压缩

好处：可以减少存储文件所需要的磁盘空间；可以加速数据在网络和磁盘上的传输。压缩格式有:DEFLATE，Gzip, bzip2, LZO等

对大文件来说，不应该使用不支持切分整个文件的压缩格式，否则将失去数据的本地特性，进而造成MapReduce应用效率低下。

如果输入文件是压缩的，那么在根据文件扩展名推断出相应codec后，MapReduce会在读取文件时自动解压缩文件

序列化

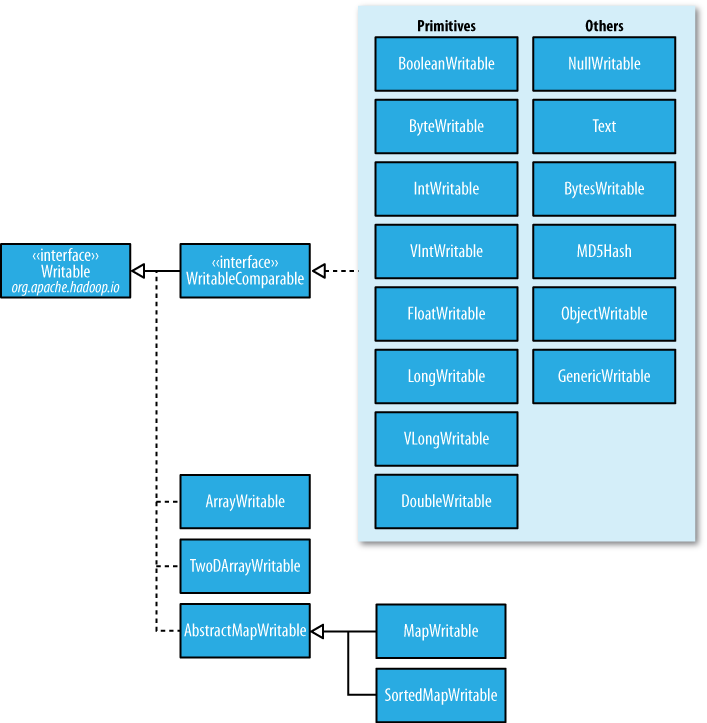
所谓序列化是指将结构化对象转化为字节流，以便以网络上传输或写到磁盘进行永久存储。反序列化是指将字节流转回结构化对象的逆过程

序列化在分布式数据处理的两大领域经常出现：进程间通信和永久存储

在Hadoop中，系统中多个节点上进程间的通信是通过“远程过程调用RPC实现的。RPC协议将消息序列化成二进制流后发送到远程节点，远程节点接着将二进制流反序列化为原始消息。

Hadoop使用自己的序列化格式Writable。

Java基本类型的Writable封装器



Apache Avro

是一个独立于编程语言的数据序列化系统，旨在解决Hadoop中Writable类型的不足：缺乏语言的可移植性。可被多语言处理的数据格式。允许其他编程语言能够读写数据，该类数据格式进行读写操作。

Avro数据是用语言无关的模式定义的。Avro模式通常用JSON编写，而数据通常用二进制格式编码

Avro为一系列对象指定一个对象容器格式—类似于Hadoop的顺序文件。Avro数据文件包含元数据项，模式数据存储在其中，这使文件可以自我声明。Avro数据文件支持压缩，并且是可切分的。

Avro数据文件 (.avro)

数据文件的头部分包含元数据，包括一个Avro模式和一个sync marker，紧接着是一系列包含序列化Avro对象的数据块。数据块由sync marker来分隔，并且允许在文件中搜索到任意位置之后通过块边界快速地重新进行同步。因此Avro数据文件是可切分的，因此适合MapReduce的快速处理。

SequenceFile （顺序文件）

提供二进制键值对的永久存储的数据结构，适合将每个二进制数据的大对象blob融入自己的文件中

MapFile

已经排序的SequenceFile，=data+index两个文件