### Hadoop核心组件

#### HDFS

**简介：**

HDFS采用主从结构模型，一个HDFS集群是由一个NameNode和若干个DataNode组成的。NameNode作为主服务器，管理文件系统命名空间和客户端对文件的访问操作。DataNode管理存储的数据。HDFS支持文件形式的数据。从内部来看，文件被分成若干个数据块，这若干个数据块存放在一组DataNode上。NameNode执行文件系统的命名空间，如打开、关闭、重命名文件或目录等，也负责数据块到具体DataNode的映射。DataNode负责处理文件系统客户端的文件读写，并在NameNode的统一调度下进行数据库的创建、删除和复制工作。NameNode是所有HDFS元数据的管理者，用户数据永远不会经过NameNode。

**体系结构：**

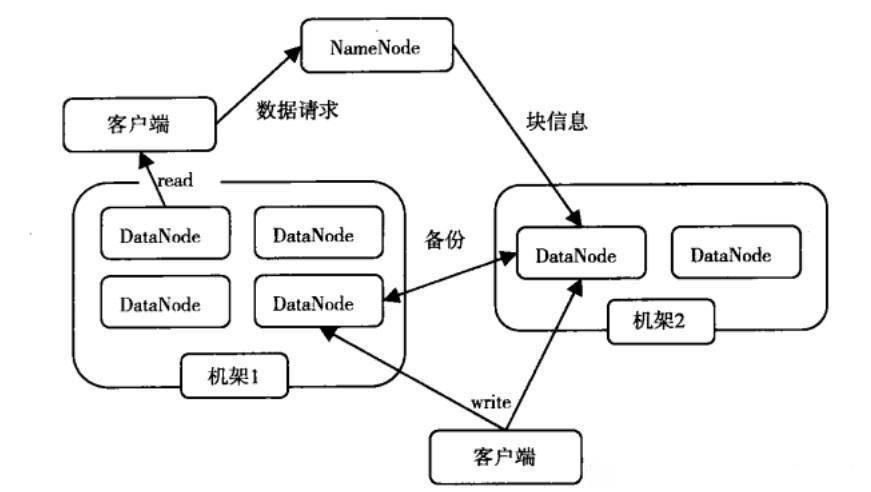


图 1 HDFS体系结构图

NameNode是管理者，DataNode是文件存储者、Client是需要获取分布式文件系统的应用程序。NameNode负责：接收用户操作请求，维护文件系统的目录结构，管理文件与Block之间关系，Block与DataNode之间关系。DataNode负责：存储文件，文件被分成Block存储在磁盘上 为保证数据安全，文件会有多个副本。

**与Google GFS的异同：**

HDFS 最早是根据 GFS（Google File System）的论文概念模型来设计实现的。

相同点：GFS和HDFS都采用单一主控机+多台工作机的模式，由一台主控机(Master)存储系统全部元数据，并实现数据的分布、复制、备份决策，主控机还实现了元数据的checkpoint和操作日志记录及回放功能。工作机存储数据，并根据主控机的指令进行数据存储、数据迁移和数据计算等。

不同点：GFS允许文件被多次或者多个客户端同时打开以追加数据，以记录为单位。而在HDFS中，HDFS文件只允许一次打开并追加数据，客户端先把所有数据写入本地的临时文件中，等到数据量达到一个Chunk的大小（通常为64MB），请求HDFS Master分配工作机及Chunk编号，将一个Chunk的数据一次性写入HDFS文件。

#### Mapreduce

**简介：**

MapReduce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采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的分片（split），这些分片可以被多个Map任务并行处理。MapReduce框架采用了Master/Slave架构，包括一个Master和若干个Slave。Master上运行JobTracker，Slave上运行TaskTracker。

**体系结构：**

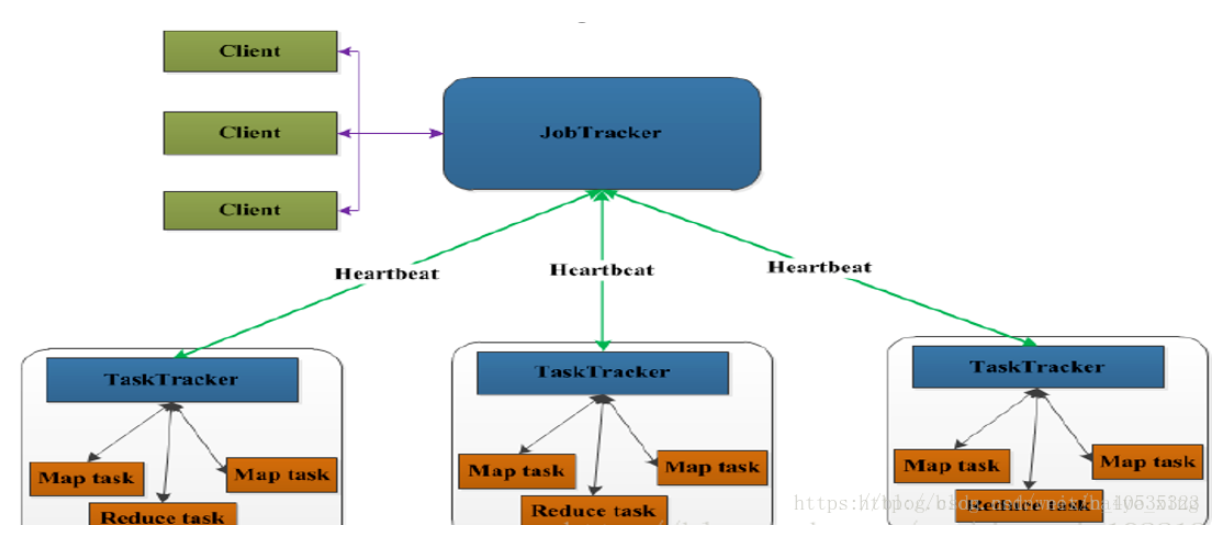


图 2 MapReduce体系结构

对于Client。可通过Client提供的一些接口查看作业运行状态。编写的MapReduce程序通过Client提交到JobTracker端。

对于JobTracker。JobTracker负责资源监控和作业调度。JobTracker 会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器，而调度器会在资源出现空闲时，选择合适的任务去使用这些资源。

对于TaskTracker。TaskTracker 会周期性地通过“心跳”将本节点上资源的使用情况和任务的运行进度汇报给JobTracker，同时接收JobTracker 发送过来的命令并执行相应的操作。 对于Task。Task 分为Map Task 和Reduce Task 两种，均由TaskTracker 启动.

MapReduce与传统并行计算呢框架相比有许多优点，具体对比如表格1所示。

表格 1 MapReduce与传统并行计算框架对比表

|  |  |  |
| --- | --- | --- |
|  | **传统并行计算框架** | **MapReduce** |
| 集群架构/容错性 | 共享式（共享内存、共享存储），容错性差 | 非共享式，容错性好 |
| 硬件/价格/扩展性 | 刀片服务器、高速网、SAN，价格贵，扩展性差 | 普通PC机，便宜，扩展性好 |
| 编程/学习难度 | what-how，难 | what，简单 |
|  | 实时、细粒度计算、计算密集型 | 批处理、非实时、数据密集型 |

#### YARN

**简介：**

YARN总体上仍然是Master/Slave结构，在整个资源管理框架中，ResourceManager为Master，NodeManager为Slave，ResourceManager负责对各个NodeManager上的资源进行统一管理和调度。当用户提交一个应用程序时，需要提供一个用以跟踪和管理这个程序的ApplicationMaster，它负责向ResourceManager申请资源，并要求NodeManger启动可以占用一定资源的任务。由于不同的ApplicationMaster被分布到不同的节点上，因此它们之间不会相互影响。

**体系结构：**

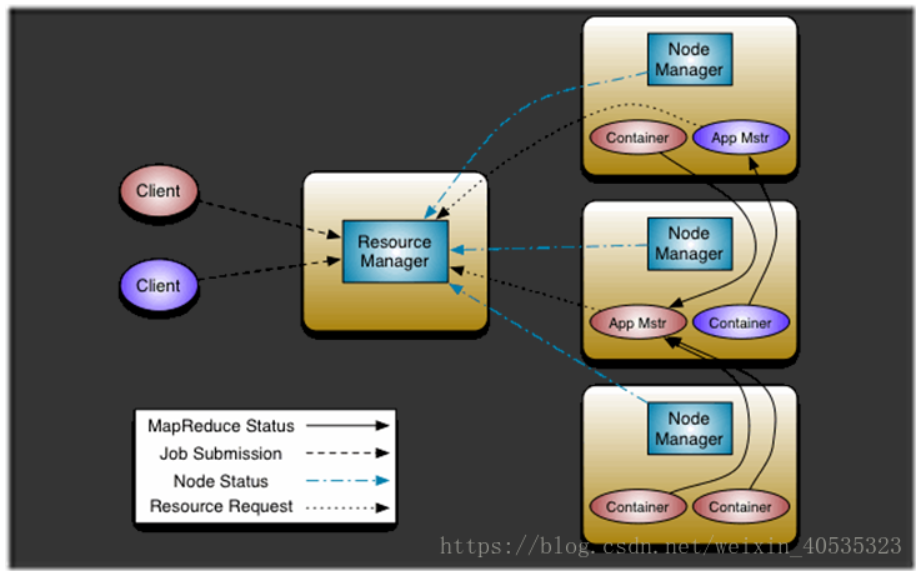


图 3 Yarn基本架构

ResourceManager是一个全局的资源管理器，负责整个系统的资源管理和分配。它主要由两个组件构成：调度器（Scheduler）和应用程序管理器（Applications Manager，ASM）。ApplicationMaster。每个作业都有一个application master来监控和管理自己的所有任务。 NodeManager是每个节点上的资源和任务管理器，一方面，它会定时地向RM汇报本节点上的资源使用情况和各个Container的运行状态；另一方面，它接收并处理来自AM的Container启动/停止等各种请求。

Container是YARN中的资源抽象，它封装了某个节点上的多维度资源，如内存、CPU、磁盘、网络等，当AM向RM申请资源时，RM为AM返回的资源便是用Container表示的。YARN会为每个任务分配一个Container，且该任务只能使用该Container中描述的资源。

### HDFS编程接口

文件的编程接口主要通过FileSystem类，目前Hadoop能够支持的文件系统如下：



#### 增加文件

接口名：public FSDataOutputStream create（Path f）throws IOException;

使用场景：创建一个Path类代表的文件，并返回一个输出流。这个方法有多个重载方法，可以用来设置是否覆盖已有文件、该文件复制的份数、写入时的缓冲区大小、文件块大小（block）、权限等。默认情况下，如果Path中文件的父目录（或者更上一级目录）不存在，这些目录会被自动创建。

#### 修改文件

1、追加内容

接口名：public FSDataOutputStream append（Path f）throws IOException;

使用场景：对已有的文件进行内容追加。

2、写入数据

接口名：public FSDataOutputStream write(buf) throws IOException;

使用场景：用于写入很多类型的数据，比如int、char、字节数组等。

应用举例：

FileSystem hdfs = FileSystem.get（new Configuration（））;

Path path = new Path（"/testfile"）;

FSDataOutputStream dos = hdfs.create（path）;

byte[] readBuf = "Hello World".getBytes（"UTF-8"）;

dos.write（readBuf, 0, readBuf.length）;

dos.close（）;

hdfs.close（）;

3、重名名

接口名：protected void rename（Path src, Path dst, Options）throws IOException;

使用场景：为指定的HDFS文件重命名。

应用举例：

Configuration conf = new Configuration（）;

FileSystem hdfs = FileSystem.get（conf）;

Path frpath = new Path（"/test"）; //旧的文件名

Path topath = new Path（"/testNew"）; //新的文件名

boolean isRename = hdfs.rename（frpath, topath）;

String result = isRename? "成功" : "失败";

#### 删除文件

接口名：public boolean delete（Path f, boolean recursive）throws IOException;

使用场景：删除指定的HDFS文件或目录（永久删除）。其中，f为需要删除文件的完整路径，recursive用来确定是否进行递归删除。如果f是一个文件或空目录，则不论recursive是何值，都删除。如果f是一个非空目录，则recursive为true时，目录下内容全部删除；如果recursive为false，不删除，并抛出IOException。

应用举例：

Path f = new Path（fileName）;

boolean isExists = hdfs.exists（f）;

if （isExists）{

boolean isDel = hdfs.delete（f, true）;

System.out.println（fileName + " delete? \t" + isDel）;

} else {

System.out.println（fileName + " exist? \t" + isExists）;

}

#### 查找文件

1、文件查找

接口名：public FileStatus[] globStatus（Path pathPattern）throws IOException;

使用场景：利用通配符来列出文件和目录。

2、文件内部查找

接口名：public long getPos（）throws IOException;

使用场景：可以查询当前在往文件的哪个位置写的写入偏移量。可以从流的任何位置读取数据。