MapReduce

基本模型

Word Count

map

```
def map(key, value):
    """
    :param key: 文件名称
    :param value: 文件内容
    """
    for w in value:
        emit(w, 1)
```

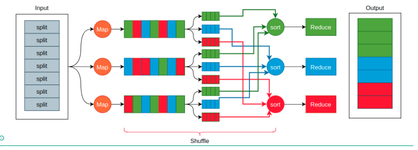
对于词频统计map函数来说，所做的映射其实就是将句子拆开，以单词为key，value固定为1

reduce

```
def reduce(key, value):
    """
    :param key: map()函数产生，为单词
    :param values: 为MapReduce框架所归约的对value集合
    """
    count = 0
    for value in values:
        count += value
    emit(count)
```

reduce()函数需要做的事情就是将MapReduce框架所归约的可迭代集合进行遍历，然后做个汇总即可

MapReduce模型



模型细节

Split阶段

假设我们并没有使用GFS或者是HDFS这类分布式存储系统来存储原始文件，而是使用了本地大文件存储，一个文件就是1TB

为了有更多的Map Worker投入计算中，对文件数据进行切分时，也需要尽可能的小 — 当然，这与集群中的节点数目、节点计算资源相关

Map阶段

在Map阶段，通常会有成百上千个Map Worker执行由用户自定义的map()函数，将文件内容转换成新的key-value对

Map端Shuffle阶段

保存临时文件

计算得到的key-value对将首先保存在Worker节点的内存缓冲区中，通常称之为环形缓冲区(从头开始写，一直写到缓冲区末尾，刷盘，然后继续从头开始写，所以称之为环形缓冲区)

- 如果对每次的计算结果均写入磁盘的话，将会严重影响整个模型的计算性能

内存缓冲区的大小有限，而Map产生的中间结果可能会超过该缓冲区大小，所以必然需要将缓冲区数据写入至磁盘中，然后对该缓冲区进行重复利用

将缓冲区的数据写入磁盘文件的过程在MapReduce模型下称之为spill，翻译为"溢写"，这个过程会产生许多的细碎文件

PS: 这和C标准I/O库执行flush操作并没有什么区别，非要搞一个"环形缓冲区"和"溢写"这种听起来很高大上的名词 =_=

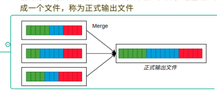
对数据进行分片

在数据写入内存环形缓冲区之前，还有一个非常重要的步骤要进行，那就是对数据进行分片。分片的目的在于当前数据到底应交于哪个Reducer进行处理，默认行为是对key进行哈希计算，然后对R进行取模

- ***hash(key) mod R***
- R值一般来说就是Reducer的数量，但是也可以由用户指定

合并Spill File

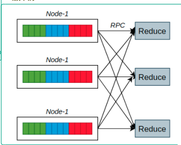
在保存临时文件阶段会产生许多的溢出文件，这些溢出文件已经按照某种分区方法进行了分区，现在需要将它们合并成一个文件，称为正式输出文件



Reduce端拉取数据

Map端所产生的正式输出文件分布在Map Worker节点的磁盘上，Reduce端需要将这些数据拷贝至自身节点内存中进行处理

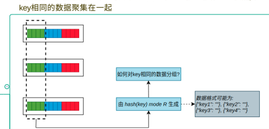
通常会使用Stream RPC的方式进行数据拉取，以降低连接建立和其它数据的网络传输开销



如果Map Worker和原始数据存储位于同一节点的话，那么该阶段就是唯一需要网络传输的地方

Reduce端Shuffle阶段

Reduce端的Shuffle阶段主要就是排序，按照正式输出文件中的key进行排序，排序的目的在于归约，也就是将所有key相同的数据聚集在一起



实现分组

使用以AVL树或红黑树为底层数据结构的字典 — 那么树结构的节点必须全部位于内存，即便有虚拟内存管理的手段存在，依然可能会导致内存溢出

使用排序的方式处理 — 完全可以使用外部排序进行，只要磁盘容量足够，多大的数据都能进行排序

Reduce阶段

此时相同的key数据已经汇总完毕，执行由用户编写的reduce()函数进行最终处理，并将结果输出至外部

角色分配

在MapReduce模型中，集群中的节点将会被赋予Mater和Worker的角色。在传统的MapReduce实现中，Mater节点为单一节点的单一运行进程实例，而Worker节点则可以有无限个

Master

- Master节点主要进行任务分发，包括Map、Reduce任务的分发
- 在Master实例中，需要记录哪些任务被分配到了哪些Worker中，执行了多长时间，执行结果如何。可以认为Master实例为记录全局任务执行状况的"最高统帅"

Worker

Worker可分为Map Worker和Reduce Worker，前者执行Map阶段，后者执行Reduce阶段

分类

Map Woker

- Map Worker的首要任务就是执行Map阶段，包括调用用户编写的map()函数，对数据进行分片，生成最终输出文件
- 其次就是需要通知Master实例当前任务已完成

Reduce Worker

- Reduce Worker的任务就是从多个节点磁盘中通过RPC的方式拉取Map Worker产生的最终输出文件，并对其进行排序、合并，并调用用户指定的reduce()函数，并将结果输出至指定位置

容错

Master实例崩溃

如果Master实例因为种种原因而崩溃，例如代码运行时错误、计算资源缺乏等，那么整个MapReduce计算过程将会直接被终止

- 单一的Master实例崩溃无法修复，同时也无法进行补偿，只能终止整个任务

Worker实例崩溃

Master节点和所有的Worker节点通过心跳连接，以检测Worker节点是否于集群中正常运行

- 当Worker节点从节点中退出，或者是Worker实例崩溃时，只需要重新将分配至该节点或者该实例的任务重新分发给其它节点或实例即可
  - 如果Worker节点挂了，那么分配至该节点的全部任务都将重新执行，无论已经执行成功的任务还是未执行的任务