Lecture 1 MAPREDUCE AND OVERVIEW OF DISTRIBUTED SYSTEMS

1. 分布式系统简介

1. 客户机、服务器、数据中心连接至互联网
2. 数据中心内部也使用分布式连接
3. 分布式系统定义：多个计算机通过网络连接，相互协作来提供服务。其中多部计算机之间只能通过数据包（packet）进行交互，这点与多处理器系统通过共享内存来进行交互不同
4. 分布式系统的作用：
    1. 用来连接物理上隔离的机器，允许用户进行信息共享
    2. 通过并行（parallelism）来提高性能。例如用于满足并行的需求
    3. 容错(tolerate faults)。当一部分服务端宕机时，可以保证始终能提供服务
    4. 通过物理上的隔离来提高安全性。例如将敏感信息放在不同的机器上以提高安全性
5. 发展历史：
    1. 随着局域网（LAN）的出现而诞生
    2. 数据中心兴起
    3. 云计算
6. 局域集群（local area clusters）
7. 分布式系统的挑战
    1. 有许多并发的组成部分
    2. 故障处理、容错等
8. 吞吐量与机器数量的关系并不直观

1.1. 组成部分

- 存储架构：将整个分布式系统的存储架构抽象为一台单个完整的计算机

- 计算架构
- 通信架构

1.2. 重要特性

1.2.1. Fault tolerance 容错

1. 在分布式系统中，由于机器数量很多，在单计算机上发生概率很低的错误可能一直在发生，因此要考虑容错。
2. 什么是容错:
    1. **Availability** 可用性: 在特定错误发生时，可以使得分布式系统继续运行。超出特定范围的错误将使系统不可继续运行。
    2. **Recoverbility** 可恢复性：在某些组件发生错误，并且进行修复后可以正常运行（在被修复之前整个系统不能工作）
3. 实现容错的方式：
    1. non-volatile storage
    2. replication
4. 研究如何使得系统高可靠，主要通过复制来实现
5. 可恢复性：使得某机器故障后恢复，并且重新参与到分布式系统中，恢复其可靠性。主要通过日志来实现

1.2.2. 一致性

同一数据可能有多个副本。因此需要考虑这些副本之间的一致性问题。

- 强一致性：通信开销很大；强制保证每个副本之间的数据是一致的。
- 弱一致性：不保证每个副本之间的数据是一致的。

1.2.3. 性能

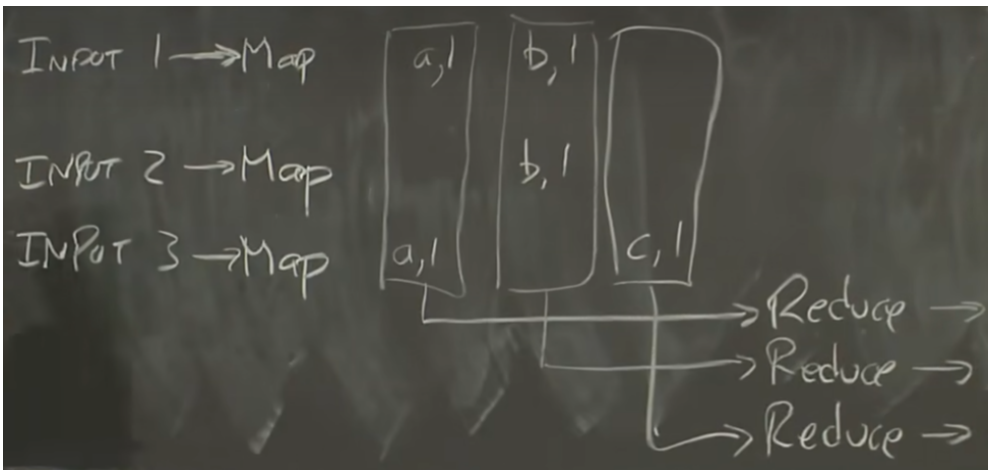
1. 性能的需求和（容错与一致性）的需求是矛盾的 。
2. 性能既包括吞吐率也包括延时（latency）。
3. 尾部延时（tail latency）

4. Scalability（可拓展性）：希望能通过n台计算机可以得到n倍的算力/吞吐量。但实际中，增加n倍的机器就得到n倍的算力这个期望并不一定能实现，因为整个系统的瓶颈可能会发生转移，不再取决于机器的数量。

---

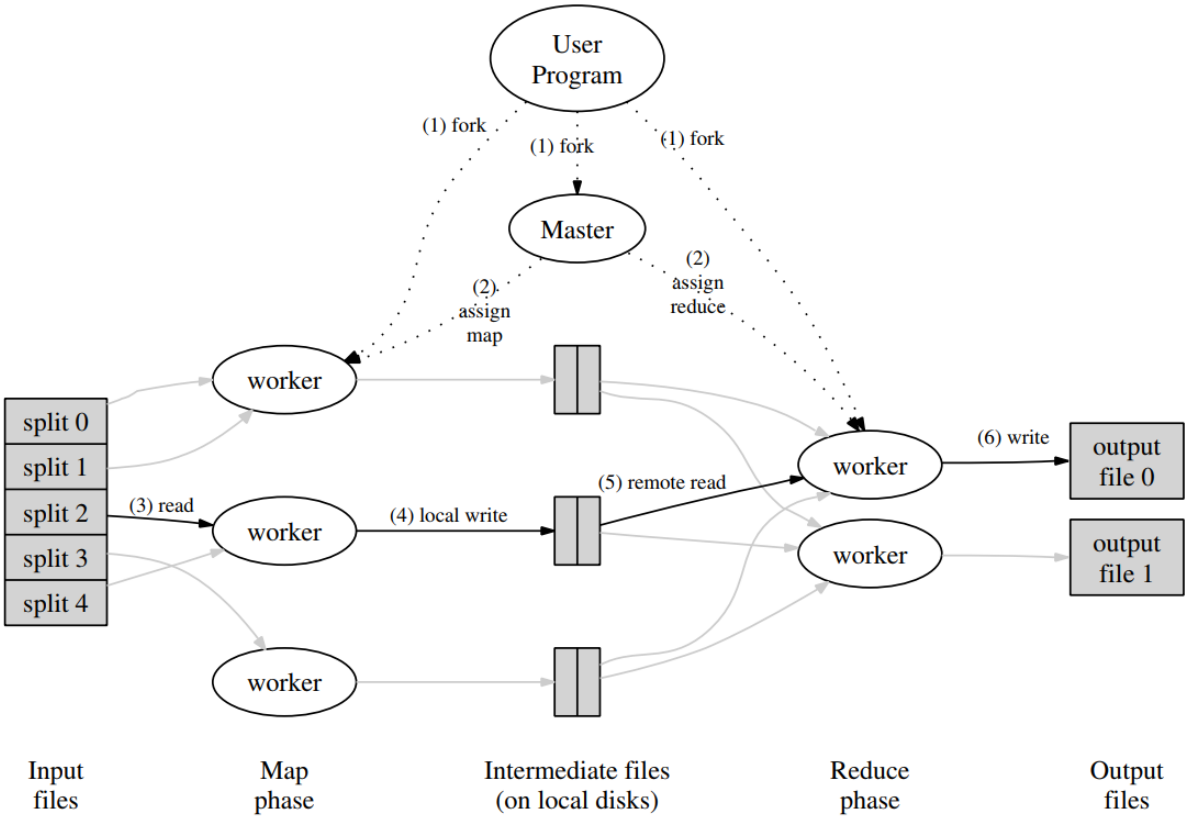
2. MapReduce

- 动机：使得编写分布式程序更加容易，隐藏底部的细节
- 方式：使用mapreduce库的map和reduce函数
- 术语（以单词数目统计为例）
    - Job：整个单词数目统计的任务
    - task：对map和reduce函数的每一次调用（invocation）
    - shuffle：本意是扑克的“洗牌，打乱次序”，在分布式计算场景中，它被引申为集群范围内跨节点、跨进程的数据分发



具体说明：在每一部机器中都是按行存储的（如上图），进行Reduce操作时，需要将按行存取转换成为按列存取（如统计单词a出现的个数），这一操作成为shuffle。shuffle必须要网络IO的参与。

2.1. 基本结构



1. map与reduce函数均由用户编写
2. 常见作用：模式匹配、URL访问次数统计、统计网站-连接图（对于某个网站，找出所有的来源）、词向量统计、建立反向索引（map统计一篇文章中的每个词，输出为<word, document ID>，而后传给reduce，reduce输出<word, document ID集合>）、分布式排序
3. Map函数：以统计单词个数为例。
    - 功能：以数据的split作为输入，以key-value作为输出（论文中被称为intermediate value）。
    - 形式：Map(k, v)，其中k为文件名， v为该文件的完整数据。
    - Map函数的例子（程序员视角）：

```
split v into words;
for each word w:
    emit(w, "1");  //对于每个单词，只有1个数目。重复单词会被emit多次
//emit函数由MapReduce框架提供。
```

    - Map函数可以避免网络IO。**在GFS中，同一个文件按照块被分布式地存储在不同的机器上。**（此时可以发现，需要读取一个文件时，可以多台机器并行读取，加大了读取时的吞吐率。题外话，与此处无关。）并且，每一台机器既作为GFS的主机，也作为一个

worker。因此，针对一个文件，一个Map函数被调用（invoke）时，存放该文件的主机同时执行Map操作（针对存在自己主机上的块），而后将中间值（intermediate value）放在本地磁盘上。

4. Reduce函数：以统计单词个数为例。
    - 功能：以key和一个vector（论文中写为迭代器，iterator）作为参数。其中这个vector由所有的Map返回值key-value对的value组成。
    - 形式：Reduce(len(v))
    - Reduce函数需要进行网络IO！

2.2. 细节

为了方便表述，可能会将运行map的worker称为mapper，运行reduce的worker称为reducer

1. 将需要操作的文件分割成为M个块（参数可由应用程序员指定），每个块通常为$16 \sim 64MB$，而后在计算机集群上运行程序的副本。
2. 特殊的程序：调度器（coordinater）/主程序（master）。调度器为workers分配tasks（map/reduce）。所有的map函数完成之后，会将中间结果通过分区函数将其划分为R个部分。注意，调度器会优先选择空闲的worker分配任务
3. map得到的中间结果存在运行map的机器内存缓冲中。这些缓冲会被定期写入磁盘。worker会将磁盘保存中间结果的位置发送给调度器。调度器会将地址转发给运行reduce的worker。某一台map完成的中间结果会被分成R部分供R个Reducer使用。
4. reducer收到调度器转发的地址后，会通过远程过程调度读取所有mapper上的中间结果（仅仅读取分配给自己的那一部分）。读取完毕后，reducer会对中间结果进行排序（若内存中无法存储则使用外排序）。
5. reducer遍历排序后的中间结果，将相同键的所有中间结果作为输入传递给reduce函数
6. reduce函数接收到参数之后会进行运算，而后将运算结果（返回值）加到当前reducer的文件后面。（至少有R个reducer，因此至少有R份输出文件）
7. 所有的map和reduce任务完成后，MapReduce唤醒用户程序。用户可以选择合并reducer的输出文件，也可以不合并（通常这些文件可以作为分布式系统的新的输入）
8. 调度器的数据结构
    1. 保存每个map和reduce **task**（注意是task不是worker）的工作状态（空闲，进行中，完成）以及id

2. 作为maper将中间结果在磁盘中的地址传递给reducer的中转。（论文中称为conduit，管道）

3. 对于完成了的map任务，需要存储其中间结果地址和大小。（调度器收到这些信息就认为maper完成了任务）

9. 初始的输入文件，以及reducer的输出文件都存储在GFS（Global File System）中

2.3. 容错

2.3.1. worker的崩溃

1. 协调器每隔一段时间ping一下worker没有收到worker反馈时，则认为worker崩溃。

2. 崩溃后，master将对应的task的状态重新设定为idle。（以便参与调度）

3. 由1可知，同一个task有可能会被做两遍。对于map操作不存在问题。而对于reduce，会产生两个结果。但由于reduce是一个函数式过程，因此两个结果相同，且最终一个会被另一个覆盖

4. 对于maper的崩溃，无论是否完成任务都必须重新进行。因为中间结果存储在maper的磁盘中，无法再得到这些数据。而对于已经完成了的reducer而言，其崩溃不需要重置任务，因为reducer会将输出文件放至GFS中，而不是保存在本地。

2.3.2. 调度器的崩溃

1. master会定期记录检查点，master发生错误时可以从检查点重新开始任务。协调器一旦故障则整个任务必须重新进行。

2.3.3. 故障时的语义

1. 任何未完成的task都会将数据存在各自的临时文件中

2. map/reduce操作绝大多数情况下都是确定性的，则与顺序执行是等价的；当这两种操作不是确定性的情况下，reduce操作产生的结果可能与多种顺序执行过程对应。例如reduce操作结果1可能与顺序执行过程1对应，而reduce操作结果2可能未必与顺序执行过程1对应，可能与另一顺序执行过程对应。这一现象产生的原因可能是由于两个reduce 任务读取了来自不同map任务执行方式的结果。

2.4. 本地化

1. 网络带宽很珍贵，MapReduce采取了一些方案来减少网络带宽的使用。

    - 机房内会有一些主机通过交换机(？)相连，尽管Google机房当时的总带宽很大，分配到每一台机器上大约只有$50Mb/s$。因此需要尽量避免网络传输。

2. GFS会将初始大文件以$64MB/block$的大小分成若干块，每个块有3个副本，存在不同的机器上。调度器会优先选择调度本地机器上存储了这些块的机器，若无法达成，则会选择调度拥有这些块的在网络上最近的机器来完成map任务。

2.5. task粒度

- master调度算法的时间复杂度：$O(M+R)$，其中M为**初始输入文件分割的份数**，R为**单个Map函数执行结果分割的份数/reducer的个数。**
- master的空间复杂度：$O(M \times R)$（why？）
- 通常情况下$M+R$大于worker的个数。可以获得更好的动态负载均衡效果。
- 由于R会与最终的结果文件分割的份数有关，因此通常只调整R，使得初始输入文件的每一块大小在$16 \sim 64MB$。而R的份数通常为worker数目的若干倍即可。

2.6. Straggler问题（掉队者问题）

- straggler：一台花了很久才完成一个Map或Reduce任务的机器。
- 原因：磁盘损坏、资源竞争等
- 解决方案：在整个MapReduce任务接近完成时，master会对当前还在运行中的task进行备份，在另一台机器上再次进行计算。当原task或副本task其中之一完成时，则该task完成。

2.7. 模型改良

- Partitioning Fuction / 分区函数：通常对关键字进行字符串哈希：$hash(key) \mod N$
- Ordering Guarantees / 排序：确保一个Map worker得到的intermediate pair是按照key排序的。可以保证输出文件也按照key值进行排序。这一特征可以提高按照键值对输出文件进行随机查询的效率或满足特定场景下的要求。
- Combiner Fuction / 合并函数：在某些情况下，Map task的结果会导致出现很多重复的key-value pair，如<key, 1>。因此可以使用combiner function对这些信息进行合并。合并的时机在intermediate value被发送给reducer之前。combiner fuction和reduce函数的写法或许很像，但两者存在区别：前者的输出为即将被发送给reducer的中间结果，而后者的输出结果为最终的输出文件。**Combiner Function**是可选项。
- Input and Output Types / 输入输出格式支持： MapReduce库自带一些输入输出格式。如文本模式，键值对序列模式等。同时可以允许用户自定义输入输出类型。

- Skip Bad Records： 有些情况下（可能是第三方库支持不佳），输入文件中的特定记录（record）会确定性地导致Map或者Reduce函数崩溃。因此，MapReduce库允许跳过这些可能会导致程序崩溃的记录（**非必选**）。
- Local Execution : 本地化执行。用于调试，可选项。
- Status Information ： 状态信息。显示资源使用，任务执行情况，崩溃的worker数目等。
- Counter ： 计数器。可以对特定事件技术。在worker回应master的ping时顺便发送给master，master汇总所有worker的计数器信息（需要去除因为backup和reexecute造成的重复信息），在MapReduce任务完成后返回给用户程序。