map reduce除了单词计数器还有哪些应用场景？

答：

全网比价：

map：从不同电商平台的数据库中获取符合用户搜索条件的多种商品信息，输出（商品，价格）

reduce：对每种商品，统计最低价格，输出（商品，最低价格）

\*\*MapReduce\*\* 是 Google 提出的一种用于处理和生成大数据集的 \*\*编程模型\*\* ，可以理解成一个 \*\*框架\*\* 。

该框架含有两个由用户来实现的接口：`map` 和 `reduce` ，可以从单词计数器例子进行理解。![[Pasted image 20221204152143.jpg]]

由用户实现的 `map` 和 `reduce` 函数的伪代码为：

```Java

map(String key, String value):

// key: document name

// value: document contents

for each word w in value:

EmitIntermediate(w,"1");

reduce(String key, Iterator values):

// key: a word

// values: a list of counts

int result = 0;

for each v in values:

result += ParseInt(v);

Emit(AsString(result));

```

该编程模型中，数据形式变换可由以下模式表示：

```

map: (k1, v1) -> list(k2, v2)

reduce: (k2, list(v2)) -> list(v3)

```

`map` 函数接收一个键值对（文件，文件内容），生成一个中间键值对集合，\_MapReduce\_ 框架会将所有共用一个键key的值value组合在一起并传递给 `reduce` 函数，`reduce` 函数接收此中间键以及该键的值的集合，将这些值合并在一起，生成一组更小的值的集合 。

由于是编程框架（思想），内部数据结构的定义是很自由很宽泛的。

参考博客：

https://www.cnblogs.com/brianleelxt/p/13231463.html

https://www.zhihu.com/question/303101438

# 实现：

\_MapReduce\_ 模型可以有多种不同的实现方式，论文主要介绍了一种在 \_Google\_ 内部广泛使用的计算环境下（通过以太网交换机连接，并由商用服务器所组成的大型集群）使用的 \_MapReduce\_ 实现 。

### 执行流程

![](https://img2020.cnblogs.com/blog/2035097/202007/2035097-20200703170502368-709259845.png)

上图为此 \_MapReduce\_ 框架实现的示意图，下文基于此图对 \_MapReduce\_ 的执行过程进行描述，描述的序号与图中的序号相对应（这部分内容论文描述比较详细，所以以翻译为主，穿插个人理解以及补充后文中的优化细节）：

1. \_MapReduce\_ 库会先把文件切分成 `M` 个片段（ 每个大小为 \_16MB~64MB\_ ），存储在 \*\*GFS 文件系统\*\* ，接着，它会在集群中启动多个 \*\*程序副本\*\* 。

2. 这些程序副本（进程）中，一个为 `master` ，剩余为 `worker` ，\_master\_ 对 \_worker\_ 进行任务分配，共有 `M` 个 \_map\_ 任务以及 `R` 个 \_reduce\_ 任务（ \_M\_ 同时为文件片段数 ， \_R\_ 由用户指定），\_master\_ 会给每个空闲的 \_worker\_ 分配一个 \_map\_ 任务或者一个 \_reduce\_ 任务 。

3. 被分配了 \_map\_ 任务的 \_worker\_ 会读取相关的输入数据片段，这些数据片段一般位于该 \_worker\_ 所在的服务器上（ \_master\_ 调度时会优先使 \_map\_ 任务执行在存储有相关输入数据的服务器上，通过这种 \*\*本地执行\*\* 的方式降低服务器间网络通信，节约网络带宽 ）。它会解析出输入数据中的 \*\*键值对\*\* ，并将它们传入用户定义的 \_Map\_ 函数中，\_Map\_ 函数所生成的 \*\*中间键值对\*\* 会被缓存在内存中 。（ 要将 \_map\_ 任务和用户定义的 \_Map\_ 函数区分开来，\_map\_ 任务包含了一些前置处理以及 \_Map\_ 函数的执行 ，\_reduce\_ 任务和 \_Reduce\_ 函数同理 ）

4. 每隔一段时间，被缓存的中间键值对会被写入到本地硬盘，并通过分区函数（一般是哈希后取模）分到 \_R\_ 个区域内 。这些被缓存的键值对在本地硬盘的位置会被传回 \_master\_ ，\_master\_ 负责将这些位置转发给执行 \_reduce\_ 任务的 \_worker\_ 。

5. 所有 \_map\_ 任务执行结束后，\_master\_ 才开始分发 \_reduce\_ 任务 。当某个执行 \_reduce\_ 任务的 \_worker\_ 从 \_master\_ 获取到了这些位置信息，该 \_worker\_ 就会通过 \*\*RPC\*\* 的方式从保存了对应缓存中间数据的 \_map workers\_ 的本地硬盘中读取数据 （ 输入一个 \_reduce\_ 任务中的中间数据会产生自所有 \_map\_ 任务 ）。当一个 \_reduce worker\_ 读完所有中间数据后，会 \*\*根据中间键进行排序，使得具有相同中间键的数据可以聚合在一起\*\* 。（需要排序是因为中间 \_key\_ 的数量一般远大于 \_R\_ ，许多不同 \_key\_ 会映射到同一个 \_reduce\_ 任务中 ）如果中间数据的数据量太大而无法放到内存中，需要使用外部排序 。

6. \_reduce worker\_ 会对排序后的中间数据进行遍历，对于每个唯一的中间键，将该中间键和对应的中间值的集合传入用户提供的 \_Reduce\_ 函数中，\_Reduce\_ 函数生成的输出会被追加到这个 \_reduce\_ 任务分区的输出文件中 （ 即一个 \_reduce\_ 任务对应一个输出文件，即 \_R\_ 个输出文件，存储在 \_GFS\_ 文件系统，需要的话可作为另一个 \_MapReduce\_ 调用的输入 ）。

7. 当所有的 \_map\_ 和 \_reduce\_ 任务完成后，\_master\_ 会唤醒用户程序 。此时，用户程序会结束对 \_MapReduce\_ 的调用 。

### 容错

#### Woker 故障

\_master\_ 会周期性地 \_ping\_ 每个 \_worker\_ ，若在一定时间内无法收到某个 \_worker\_ 的响应，那么 \_master\_ 将该 \_worker\_ 标记为 \_fail\_ ：

- 此 \_worker\_ 上 \*\*完成\*\* 的所有 \_map\_ 任务都被重设为 \_idle\_ 状态，交由别的 \_worker\_ 去执行这些 \_map\_ 任务

- 此 \_worker\_ 上 \*\*正在执行\*\* 的 \_map\_ 任务或 \_reduce\_ 任务重设为 \_idle\_ 状态，并等待重新调度

该 \_worker\_ 上完成的 \_map\_ 任务必须重新执行，因为 \_map\_ 任务数据结果保存在 \_worker\_ 的本地硬盘中，\_worker\_ 无法访问了，则输出数据也无法访问；该 \_worker\_ 上完成的 \_reduce\_ 任务不需要重新执行，因为输出结果已存储在全局文件系统中 。

#### Master 故障

目前的实现选择中断 \_MapReduce\_ 计算，客户端可检查该 \_master\_ 的状态，并根据需要重新执行 \_MapReduce\_ 操作 。

### 数据存储位置

此模式是为了 \*\*节约网络带宽\*\* 。

将输入数据（ 由 \_GFS\_ 系统管理 ）存储在集群中服务器的本地硬盘上，\_GFS\_ 将每个文件分割为大小为 \_64MB\_ 的 \_Block\_ ，并且对每个 \_Block\_ 保存多个副本（通常3个副本，分散在不同机器上）。\_master\_ 调度 \_map\_ 任务时会考虑输入数据文件的位置信息，尽量在包含该相关输入数据的拷贝的机器上执行 \_map\_ 任务 。若任务失败，\_master\_ 尝试在保存输入数据副本的邻近机器上执行 \_map\_ 任务，以此来节约网络带宽 。

### 备用任务

此模式是为了缓解 \*\*straggler (掉队者) 问题\*\* ，即 ：一台机器花费了异常多的时间去完成 \*\*最后几个\*\* \_map\_ 或 \_reduce\_ 任务，导致整个计算时间延长的问题 。可能是由于硬盘问题，可能是 \_CPU\_ 、内存、硬盘和网络带宽的竞争而导致的 。

解决此问题的方法是：当一个 \_MapReduce\_ 计算 \*\*接近完成\*\* 时，\_master\_ \*\*为正在执行中的任务执行 备用任务\*\* ，当此任务完成时，无论是主任务还是备用任务完成的，都将此任务标记为完成 。这种方法虽然多使用了一些计算资源，但是有效降低了 \_MapReduce Job\_ 的执行时间 。

### Combiner 函数

某些情况下，每个 \_map\_ 任务生成的中间 \_key\_ 会有明显重复，可使用 \*\*Combiner 函数\*\* 在 \_map worker\_ 上将数据进行部分合并，再传往 \_reduce worker\_ 。

\_Combiner\_ 函数 和 \_Reduce\_ 函数的实现代码一样，区别在于两个函数输出不同，\_Combiner\_ 函数的输出被写入中间文件，\_Reduce\_ 函数的输出被写入最终输出文件 。

这种方法可以提升某些类型的 \_MapReduce\_ 任务的执行速度（ 如 \_word count\_ 任务）。

### 临时中间文件

对于有服务器故障而可能导致的 \*\*reduce 任务可能读到部分写入的中间文件\*\* 的问题 。可以使用 \*\*临时中间文件\*\* ，即 \_map\_ 任务将运算结果写入临时中间文件，一旦该文件完全生成完毕，以原子的方式对该文件重命名 。

要实现的话，感觉欠缺很多前置知识。

https://pdos.csail.mit.edu/6.824/labs/lab-mr.html

这是lab1的说明

跟jyy的形式差不多，也有一个测试脚本，也有git，也有例子文件。但是是GO

实现一个worker和master

https://github.com/OneSizeFitsQuorum/MIT6.824-2021/blob/master/docs/lab1.md

这里有一个go的mapreduce lab1的实现

看不懂啊

hadoop是一个mapreduce的实现

csdn上有很多实际应用hadoop mapreduce的例子，以及对于hadoop mapreduce执行步骤的解析

https://zh.wikipedia.org/wiki/MapReduce

也就是说，程序员实现map和reduce函数

然后，mapreduce框架进行调用处理，完成分布式的操作

可以通过对于应用hadoop的例子，了解实现mapreduce框架需要实现哪些函数

大概也就worker和master两个？