MapReduce 编程简介

December 18, 2020

MapReduce

MapReduce [1] 是一种编程模型，一种能够处理和生成大型数据集的关联实现。

用户只需要编写 map 函数和 reduce 函数，而不用考虑并行化、容错、数据分布、负载均衡等一系列的问题。

- map 函数，将原始数据的一个 key/value 对，转换为一组中间 key/value 对。
- reduce 函数，将 map 函数生成的中间 key/value 对，对 value 按照同样的 key 进行合并。

具体数学形式为：

$$\begin{cases} map & (k1, v1) & \rightarrow list(k2, v2) \\ reduce & (k2, list(v2)) & \rightarrow list(v2) \end{cases}$$

MapReduce 执行流程

- 1. MapReduce 用户库，先将原始数据切分为 M 块，每一块大致为 64M 大小。然后，在计算机集群上启动许多的程序副本。

MapReduce 执行流程

- 1. MapReduce 用户库，先将原始数据切分为 M 块，每一块大致为 64M 大小。然后，在计算机集群上启动许多的程序副本。
- 2. 其中有一个副本程序是特殊的，被叫做 master。其他的副本程序叫做 workers，用来执行 master 分配的 map 和 reduce 任务。总共有 M 个 map 任务，R 个 reduce 任务。master 选择一些空闲的 workers，给每一个 worker 分配一个 map 任务或者一个 reduce 任务。

MapReduce 执行流程

- 1. MapReduce 用户库，先将原始数据切分为 M 块，每一块大致为 64M 大小。然后，在计算机集群上启动许多的程序副本。
- 2. 其中有一个副本程序是特殊的，被叫做 master。其他的副本程序叫做 workers，用来执行 master 分配的 map 和 reduce 任务。总共有 M 个 map 任务，R 个 reduce 任务。master 选择一些空闲的 workers，给每一个 worker 分配一个 map 任务或者一个 reduce 任务。
- 3. 一个被分配 map 任务的 worker，开始读取输入的数据块。从输入数据块中解析 key/value 对，将其传入到用户定义的 map 函数中。这个产生的中间 key/value 对集合暂时缓存在 worker 的内存中。

MapReduce 执行流程

- 4. worker 周期性地，将缓存的中间 key/value 对集合，写入本地硬盘。并且，通过一个分区函数 (比如，hash(key) mod R) 将数据划为到不同的区域。这些在 worker 本地磁盘的中间 key/value 对集合的文件位置信息，被发送到 master 上。再由 master 将文件位置信息传输到 reduce 任务的 workers 上。

MapReduce 执行流程

- 4. worker 周期性地，将缓存的中间 key/value 对集合，写入本地硬盘。并且，通过一个分区函数 (比如，hash(key) mod R) 将数据划为到不同的区域。这些在 worker 本地磁盘的中间 key/value 对集合的文件位置信息，被发送到 master 上。再由 master 将文件位置信息传输到 reduce 任务的 workers 上。
- 5. 当一个执行 reduce 任务的 worker，被 master 通知中间 key/value 对集合的位置信息时，它使用 RPC 将远程的 map 任务生成的数据，传输到本机上。当 worker 拥有所有的中间 key/value 对集合数据时，它会根据 key 进行排序数据，以便让相同的 key 分组在一起。

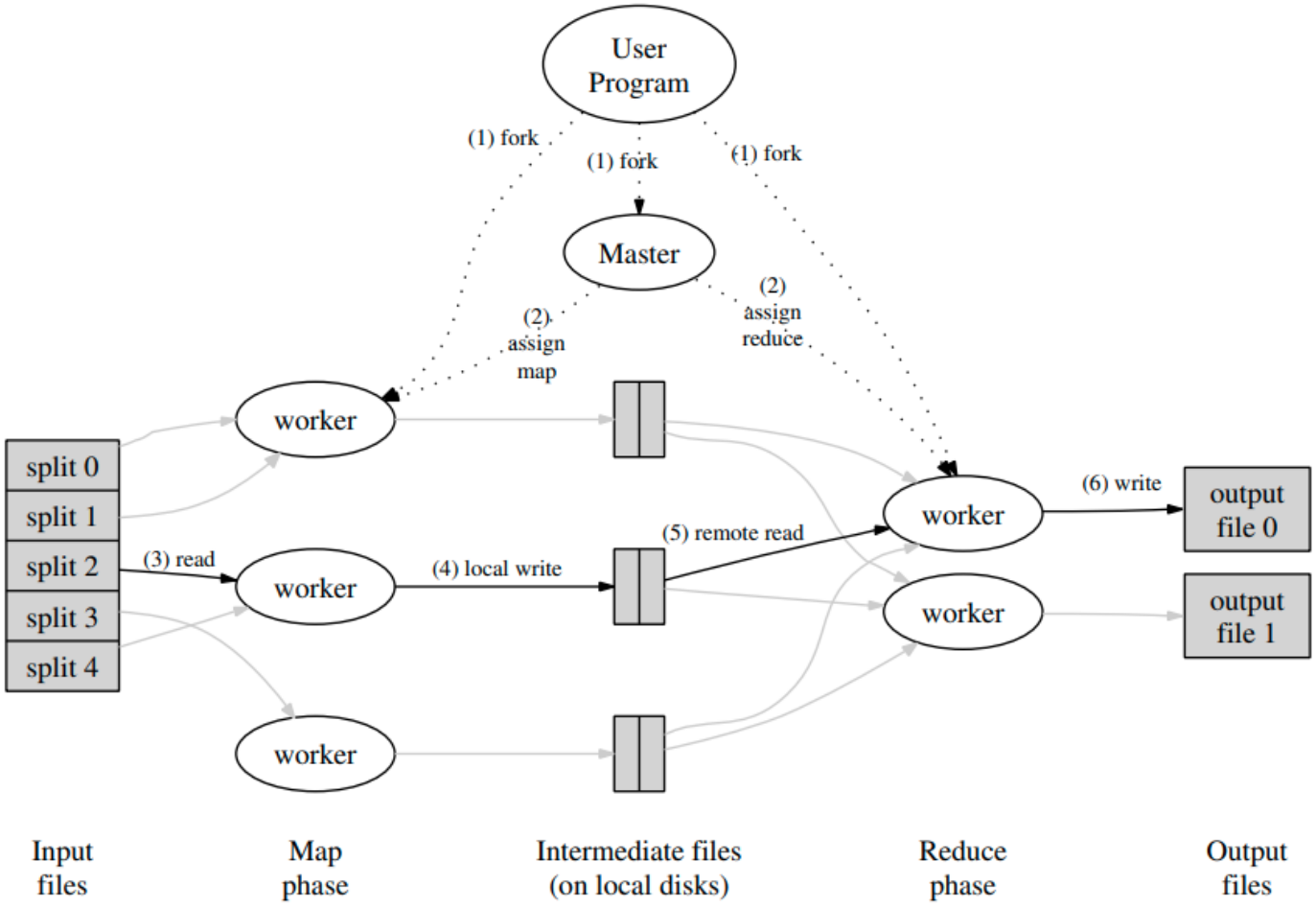
MapReduce 执行流程

- 6. reduce 任务的 worker，通过对拥有相同 key 的 value 集合进行遍历，将遍历的值 (key,list(values)) 传入到用户定义的 reduce 函数中，reduce 函数的结果被追加到当前 reduce partition 的文件中。

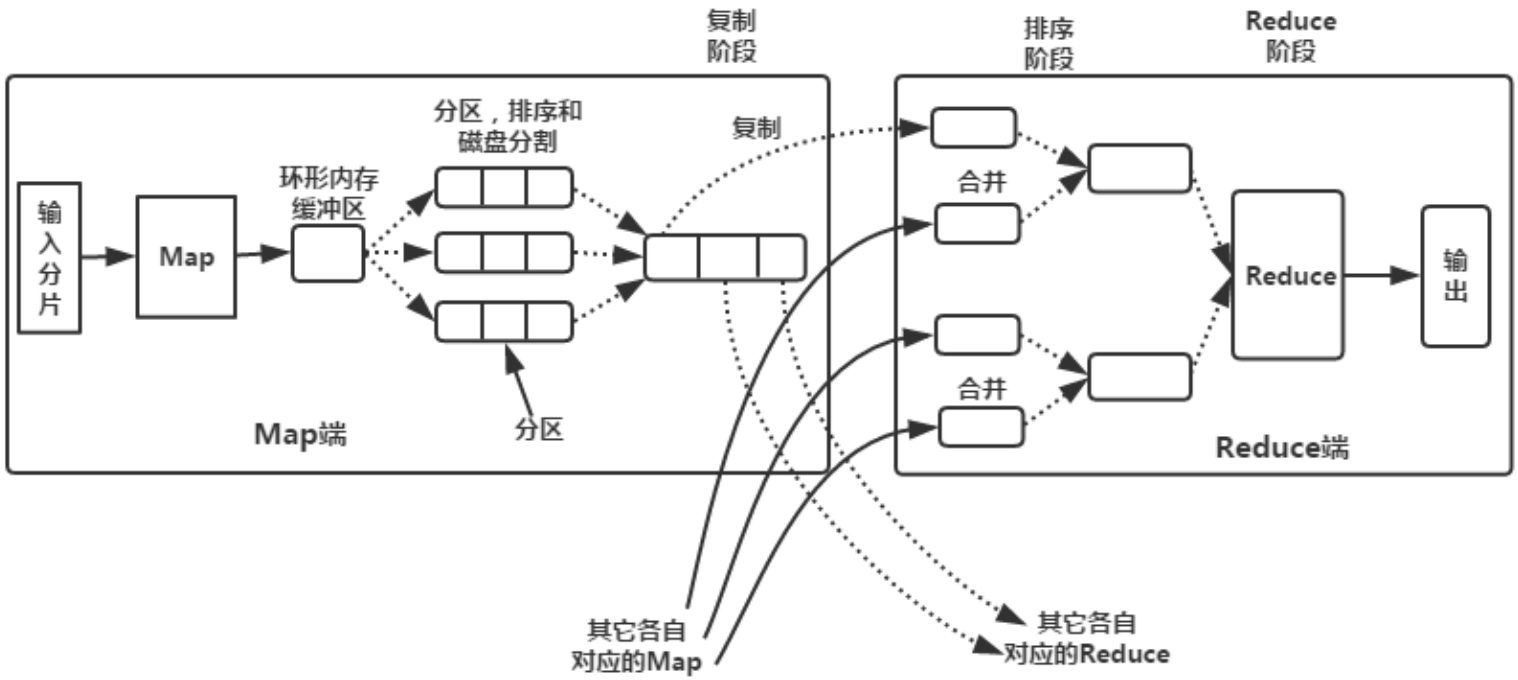
MapReduce 执行流程

- 6. reduce 任务的 worker，通过对拥有相同 key 的 value 集合进行遍历，将遍历的值 (key,list(values)) 传入到用户定义的 reduce 函数中，reduce 函数的结果被追加到当前 reduce partition 的文件中。
- 7. 当所有的 map 任务和 reduce 任务完成时，整个任务完成。唤醒 mapreduce 客户端，表明任务完成。

MapReduce 执行流程



MapReduce 执行流程



MapReduce 其他

分区函数 (partitioning Function)

mapreduce 的默认的分区函数是 HashPartition。
也可以自定义 Partition 函数，函数接口为：

```
public abstract class Partitioner<K,V> {
  public abstract int getPartition(K key,
                    V val, int numPartitions);
}
```

MapReduce 其他

分区顺序保障 (Ordering Guarantees)

在一个分区 (partition) 中，中间生成的 key/value 对集合数据总是以 key 升序被 reduce 任务处理。

MapReduce 其他

结合函数 (Combiner Function)

如果用户的 reduce 函数满足交换律和结合律，例如：

```
max(x,y,z) = max(x, max(y,z)) = max(max(x,y), z)
```

那么就可以使用可选的结合函数 (Combiner Function)。结合函数是在 map 任务的 worker 上运行的，内容和 reduce 任务是一样的。主要是对当前 map 任务的每个分区进行部分合并数据操作，减少从 map 任务的 worker 传输到 reduce 任务的 worker 的数据量，加快执行效率。

MapReduce 其他

输入和输出类型 (Input and Output Types)

hadoop mapreduce 默认的 InputFormat 是 TextInputFormat，用于解析数据块的记录。其中 value 是 Text 类型，存储一行输入；其中 Key 是 LongWritable 类型，存储该行在整个文件中的字节偏移量（不是行数)。
hadoop mapreduce 的输入输出类型有：Text 类，IntWritable 类，LongWritable 类等。

- Text 类，主要使用 UTF-8 编码存储字符串。用于操作 UTF-8 字符串，包括序列化，反序列化，字符串写入，字节比较，字节长度获取等功能。
- IntWritable 类，用于 32 位整数的序列化和反序列化，数值写入。
- LongWritable 类，用于 64 位整数的序列化和反序列化，数值写入。

一个分布式 grep 例子

map 任务

```
/**
 * (K,Text) 原始输入的key/value 对
 * (Text,LongWritable) map任务处理后的key/value 对
 **/
 public class  RegexMapper<K> extends Mapper<K, Text, Text, LongWritable> {

   private static final reg = "(hello)";
   private Patten patten;
   public void setup(Context contex) {
      patten = Patten.compile(reg);
   }
   @Override
   public void map(K key,Text value,Context context) throws IOException,
                                                        InterruptedException {
      String text = value.toString();
      Matcher matcher = patten.matcher(text);
      while(matcher.find()) {
        context.write(new Text(matcher.group(0)), new LongWritable(1));
      }
   }
 }
```

一个分布式 grep 例子

reduce 任务

```
/**
 * (K, Iterable<LongWritable>) 输入的key/value对
 * (K, LongWritable) reduce任务合并输出的key/value对
 **/
public class LongSumReducer<K> extends Reducer<K, LongWritable, K, LongWritable> {

   private LongWritable result = new LongWritable();
   @override
   public void reduce(K key,Iterable<LongWritable> values, Context context) throws
                                                    IOException, InterruptedException {
      long sum = 0;
      #合并
      for(LongWritable val: values){
          sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
  }
}
```

一个分布式 grep 例子

job 设置，提交 job

```
public class Grep {
   public static void main(String... args) throws Exception {
     Configuration conf = new Configuration();
     Job job = Job.getInstance(conf, "grep");
     job.setJarByName(Grep.class);
     #设置map任务
     job.setMapperClass(RegexMapper.class);
     #设置combiner任务
     job.setCombinerClass(LongSumReducer.class);
     #设置reduce任务
     job.setReducerClass(LongSumReducer.class);
     #设置任务输出key type 和value type
     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(LongWritable.class);
     #设置输入文件路径，默认解析器
     FileInputFormat.addInputPath(job, new Path(args[0]));
     #设置输出文件路径
     FileOutputFormat.addOutputPath(job, new Path(args[1]));
     #开启job，等待job完成
     job.waitForCompletion(true);
   }
}
```

参考

[1] Jeffrey Dean and Sanjay Ghemawat.
Mapreduce: simplified data processing on large clusters.
Communications of the ACM, 51(1):107–113, 2008.