大数据技术基础

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

第二章 大数据Hadoop —MapReduce框架

鄂海红 计算机学院 教授
ehaihong@bupt.edu.cn 微信: 87837001 QQ: 3027581960

大数据处理—MapReduce框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

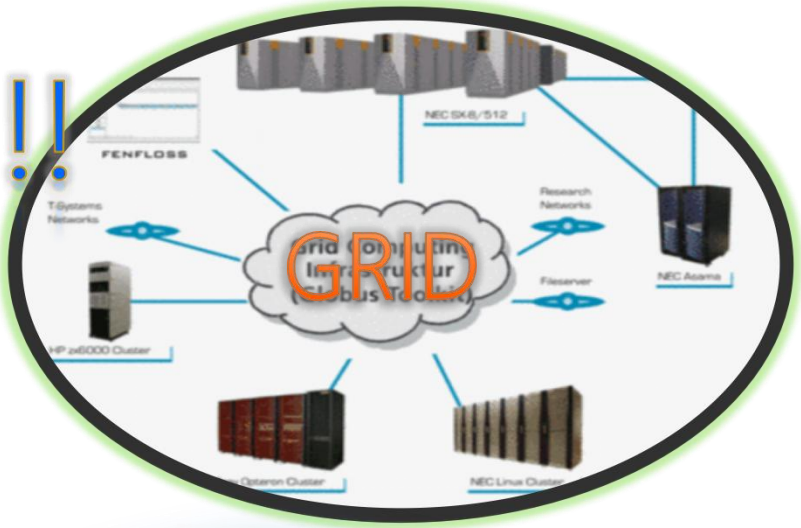
大数据处理—MapReduce框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

MapReduce的并行计算

并行计算大势所趋

并行计算!!!



海量数据并行处理技术

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 海量数据及其处理已经成为现实世界的急迫需求
- 处理数据的能力大幅落后于数据增长，需要寻求更为有效的数据密集型并行计算方法
  - 磁盘容量增长远远快过存储访问带宽和延迟：80年代中期数10MB到今天1-2TB，增长10万倍，而延迟仅提高2倍，带宽仅提高50倍！

100TB数据顺序读一遍需要多少时间?

设硬盘读取访问速率128MB/秒

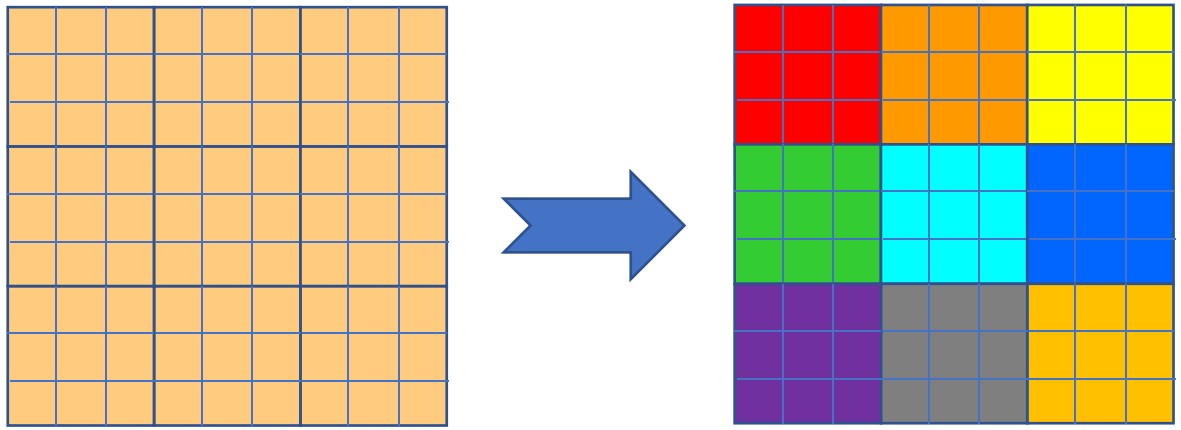
1TB/128MB 约2.17小时

100TB/128MB = 217小时 = 9天！

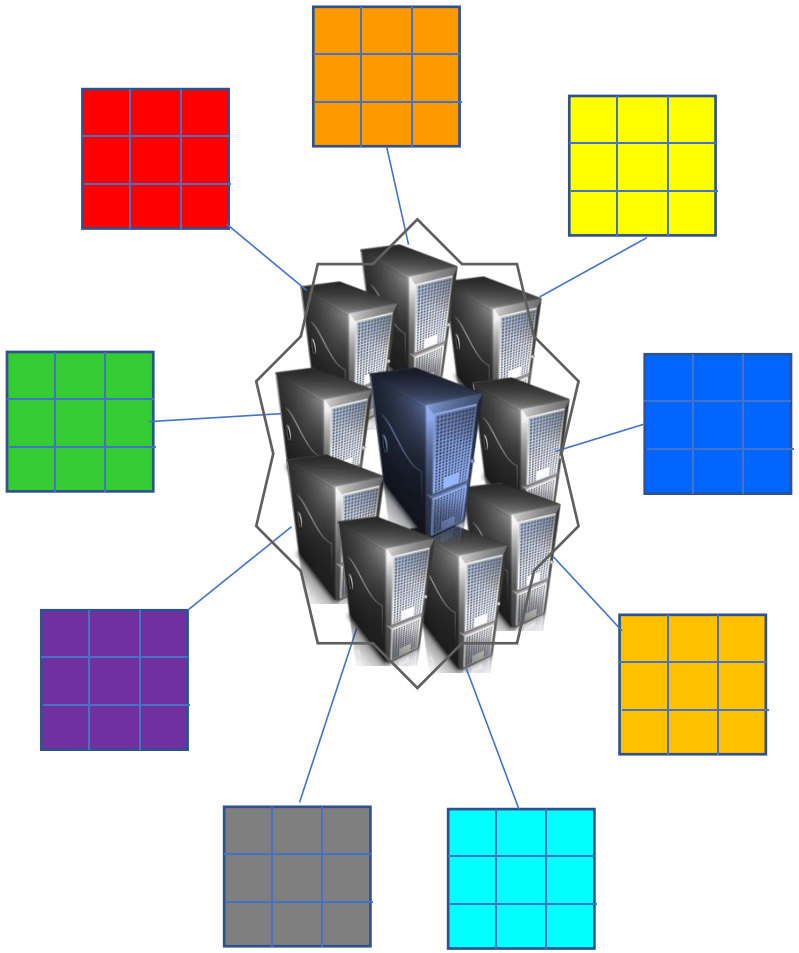
即使用百万元高速磁盘阵列(800MB/s),仍需1.5天！

大数据的并行化计算

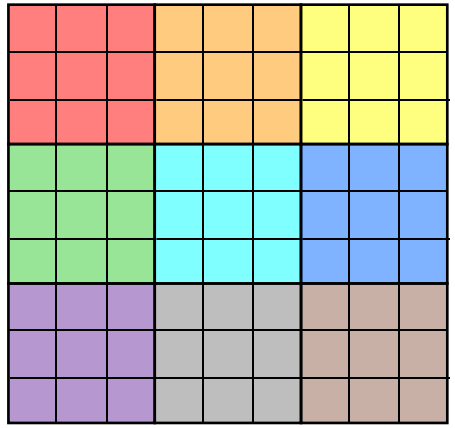
- 一个大数据若可以分为具有同样计算过程的数据块，并且这些数据块之间不存在数据依赖关系，则提高处理速度的最好办法就是并行计算
- 例如：假设有一个巨大的2维数据需要处理(比如求每个元素的开立方)，其中对每个元素的处理是相同的,并且数据元素间不存在数据依赖关系,可以考虑不同的划分方法将其划分为子数组,由一组处理器并行处理



大数据的并行化计算



合并

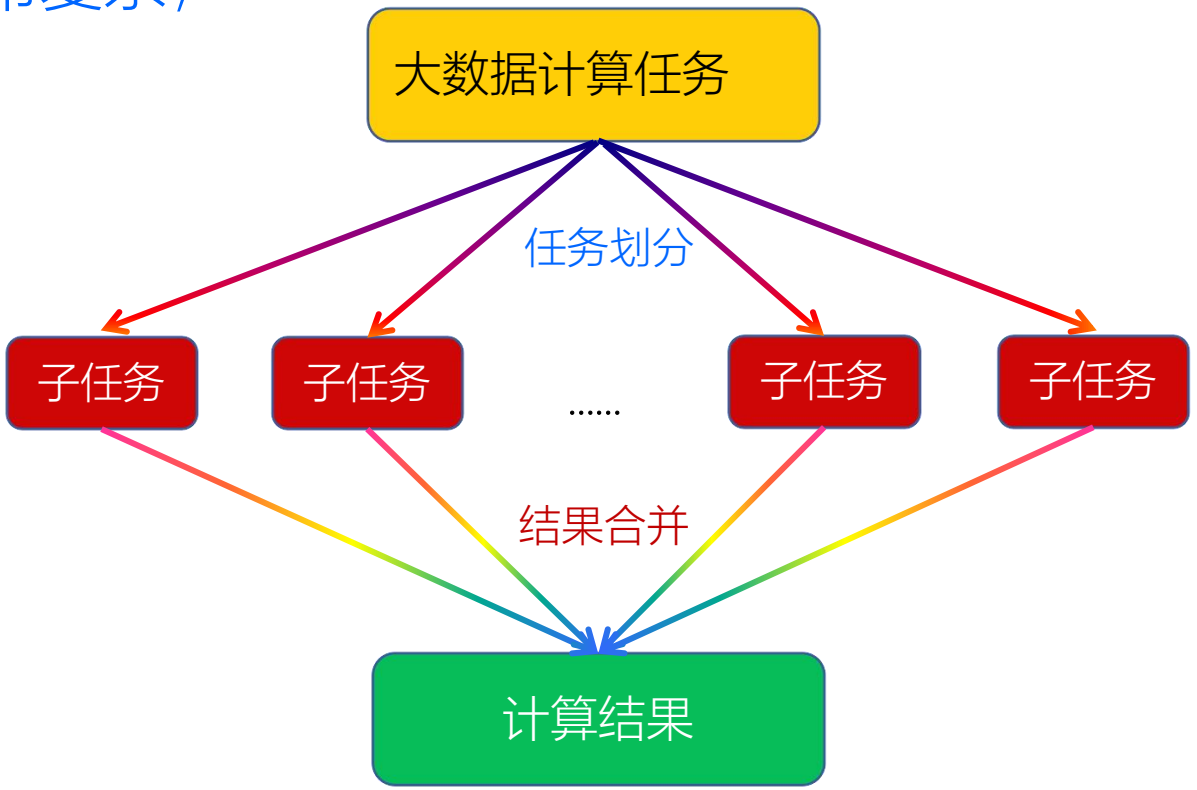


Master：负责划分和分配任务

Worker：负责数据块计算

大数据任务划分和并行计算模型

- 并行计算技术和并行程序设计的非常复杂，要考虑：
  - 不同类型的计算问题
  - 数据特征
  - 计算要求
  - 系统构架
- 程序设计需要考虑：
  ① 数据怎么划分？
  ② 计算任务和算法怎么划分？
  ③ 数据访问和通信同步怎么控制？



软件开发难度大,难以找到统一和易于使用的计算框架和编程模型与工具

为什么需要MapReduce?

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 海量数据处理需要有效的并行处理技术
  - 如何对巨量的Web文档建立索引、根据网页链接计算网页排名
  - 从上百万文档中训练垃圾邮件过滤器
  - 数十亿字符串的排序
- 解决方案：
  - 编写程序完成这些巨量数据的处理问题，MapReduce提供了一个分布式计算环境和构架
  - 让程序员仅需关注问题本身，编写很少的程序代码即可完成看似难以完成的任务
- MapReduce
  - MapReduce是一种面向大规模海量数据处理的高性能并行计算平台和软件编程框架
  - 广泛应用于搜索引擎（文档倒排索引，网页链接图分析与页面排序等）、Web日志分析、文档分析处理、机器学习、机器翻译等各种大规模数据并行计算应用领域
- MapReduce是面向海量数据处理非常成功的技术
  - MapReduce出现后，经过业界和学界实践，被验证是有效和易于使用的海量数据并行处理技术
  - 后续被Google,Yahoo,IBM,Amazon,百度、淘宝、腾讯等国内外公司实践使用

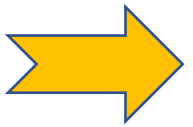
MapReduce-面向大规模数据并行处理

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 基于集群的高性能并行计算平台(Cluster Infrastructure)
  - 允许用市场上现成的普通PC或性能较高的刀架或机架式服务器，构成一个包含数千个节点的分布式并行计算集群
- 并行程序开发与运行框架(Software Framework)
  - 提供了一个庞大但设计精良的并行计算软件构架，能自动完成计算任务的并行化处理
  - 自动划分计算数据和计算任务，在集群节点上自动分配和执行子任务以及收集计算结果
  - 将数据分布存储、数据通信、容错处理等并行计算中的很多复杂细节交由系统负责处理，大大减少了软件开发人员的负担
- 并行程序设计模型与方法(Programming Model & Methodology)
  - 借助于函数式语言中的设计思想，提供了一种简便的并行程序设计方法
  - 用Map和Reduce两个函数编程实现基本的并行计算任务，提供了完整的并行编程接口，完成大规模数据处理

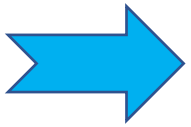
典型的流式大数据问题的特征

- 大量数据记录/元素进行重复处理
- 对每个数据记录/元素作感兴趣的处理、获取感兴趣的中间结果信息



Map

- 排序和整理中间结果以利后续处理
- 收集整理中间结果
- 产生最终结果输出



Reduce

关键思想：为大数据处理过程中的两个主要处理操作提供一种抽象机制

Map & Reduce

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- MapReduce定义了Map和Reduce两个抽象的编程接口，由用户去编程实现:
- Map: (k1; v1) → [(k2; v2)]
  - 输入(k1; v1) ： (line, “the weather is good”)
  - 处理:

```
// 根据空格将这一行切分成单词
String[] words = line.split(" ");
// 将单词输出为<单词，1>
for(String word:words){
// 将单词作为key，将次数1作为value
context.write(new Text(word), new IntWritable(1));
```

  - 输出 [(k2; v2)]： (the, 1), (weather, 1), (is, 1), (good, 1)

Map & Reduce

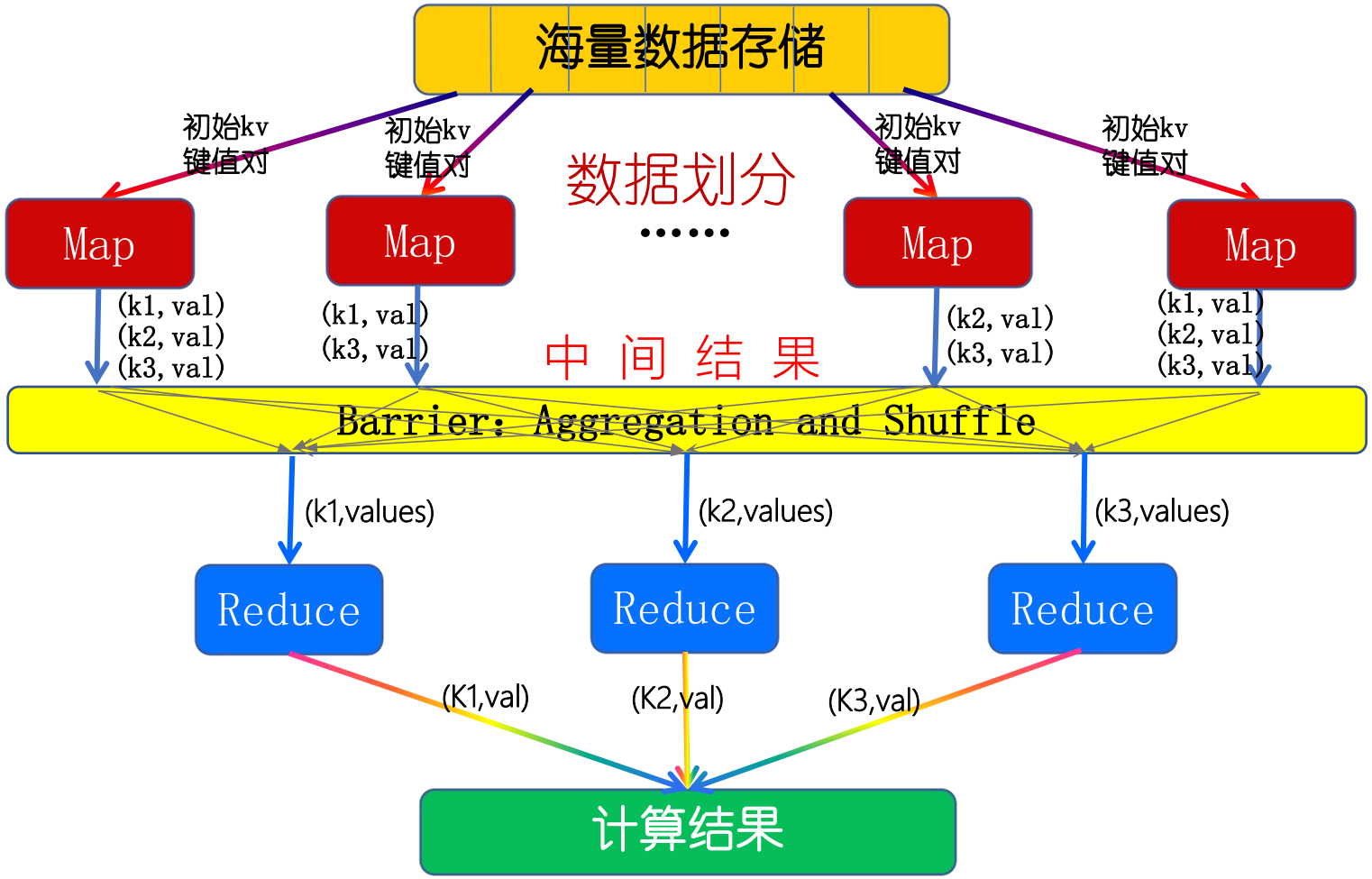
北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- reduce: (k2; [v2]) → [(k3; v3)]
  - 输入：(good, 1), (good, 1), (good, 1), (good, 1), (good, 1)
  - 处理：

```
int count = 0;
// 汇总各个key的个数
for(IntWritable value:values){
        count +=value.get(); }
// 输出该key的总次数
context.write(key, new IntWritable(count));
```

  - 输出：(good, 5)
- Map和Reduce为程序员提供了一个清晰的操作接口抽象描述

基于Map和Reduce的并行计算模型



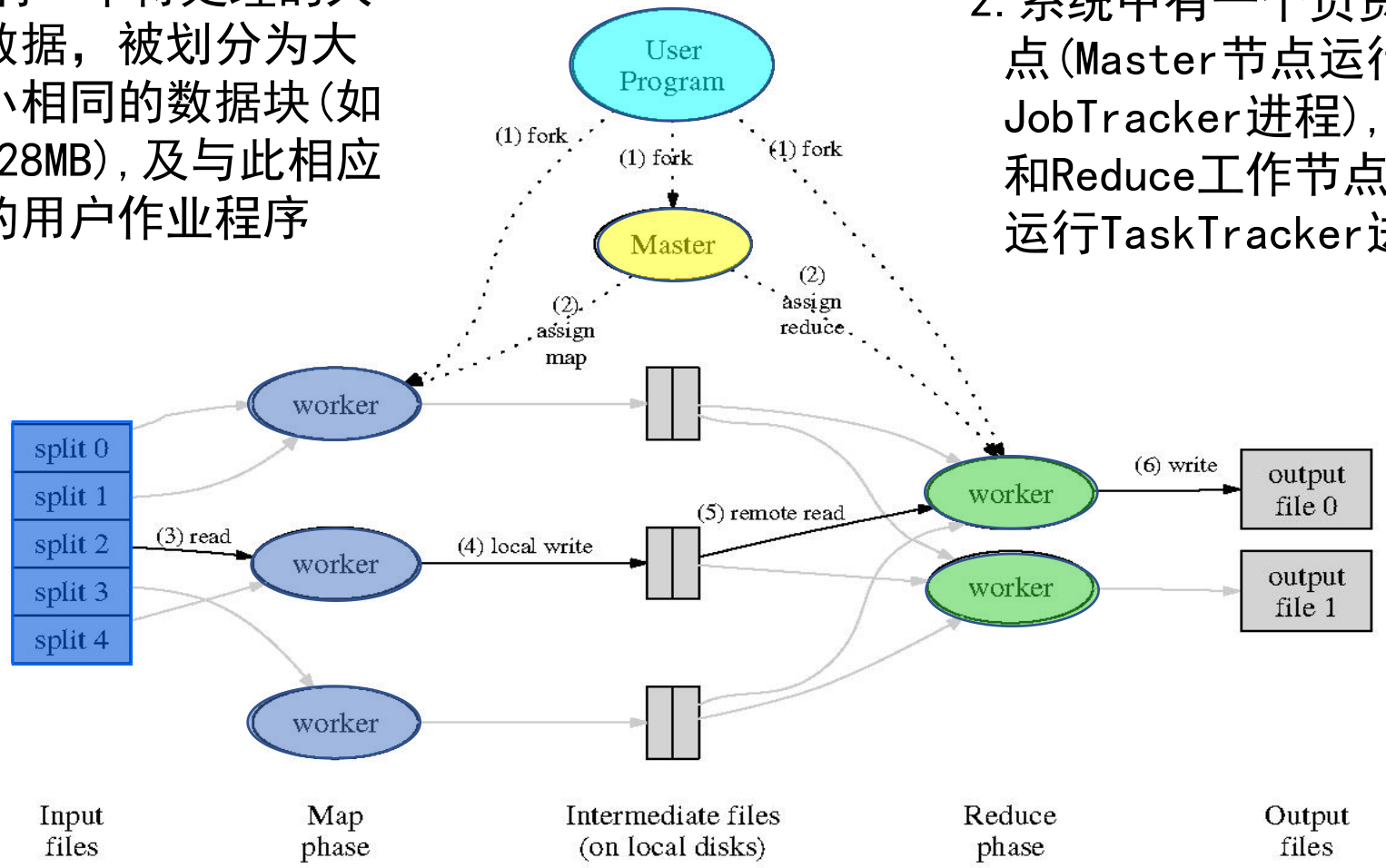
基于Map和Reduce的并行计算模型

- 各个map函数对所划分的数据并行处理，从不同的输入数据产生不同的中间结果输出
- 各个reduce也各自并行计算，各自负责处理不同的中间结果数据集合
- 进行reduce处理之前,必须等到所有的map函数做完，因此,在进入reduce前需要有一个同步障(barrier);这个阶段也负责对map的中间结果数据进行收集整理(aggregation & shuffle)处理,以便reduce更有效地计算最终结果
- 最终汇总所有reduce的输出结果即可获得最终结果

MapReduce并行处理的基本过程

1. 有一个待处理的大数据，被划分为大小相同的数据块(如128MB)，及与此相应的用户作业程序

2. 系统中有一个负责调度的主节点(Master节点运行着JobTracker进程)，以及数据Map和Reduce工作节点(Worker节点运行TaskTracker进程)



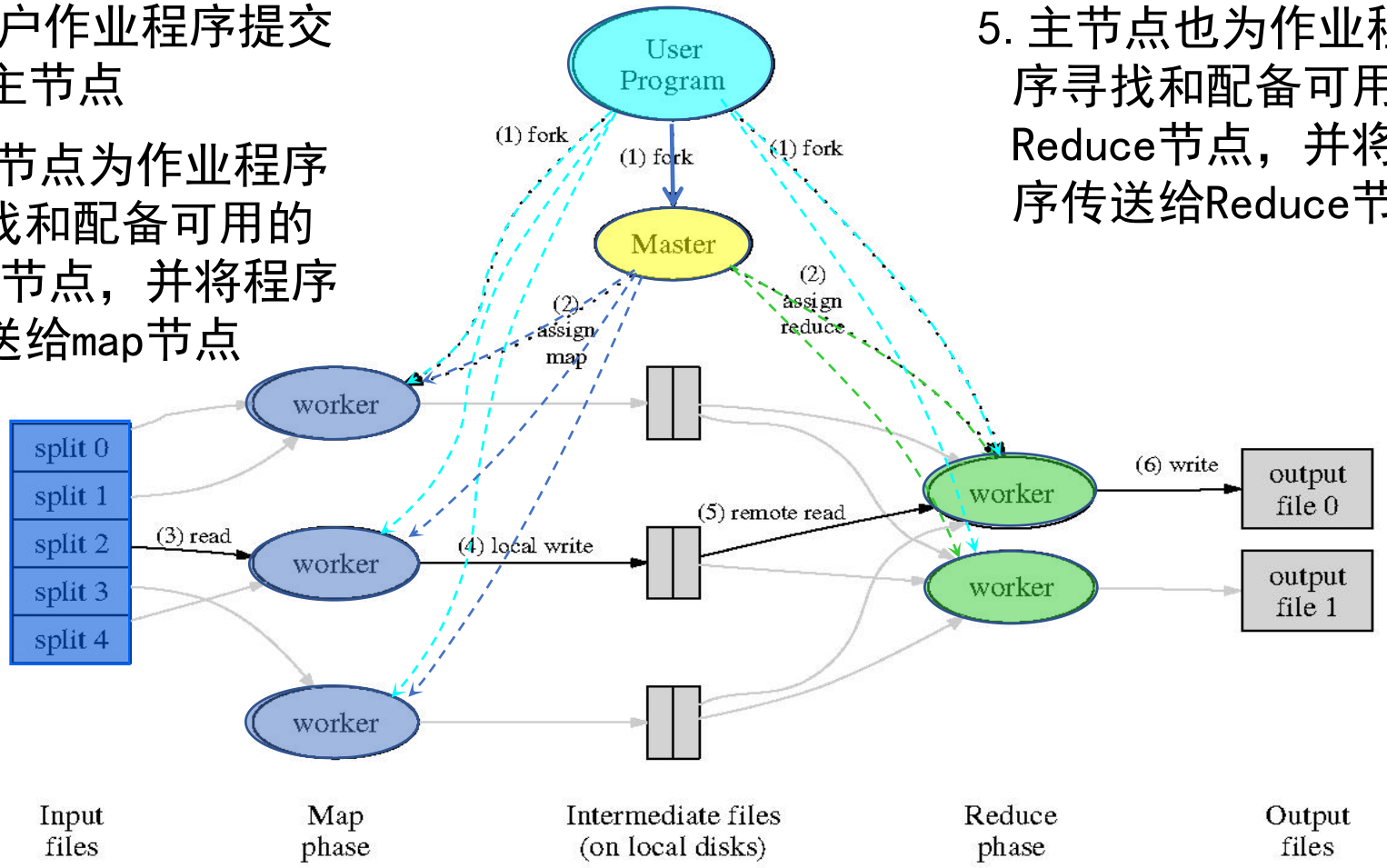
Cite from Dean and Ghemawat (OSDI 2004)

MapReduce并行处理的基本过程

3. 用户作业程序提交给主节点

4. 主节点为作业程序寻找和配备可用的Map节点，并将程序传送给map节点

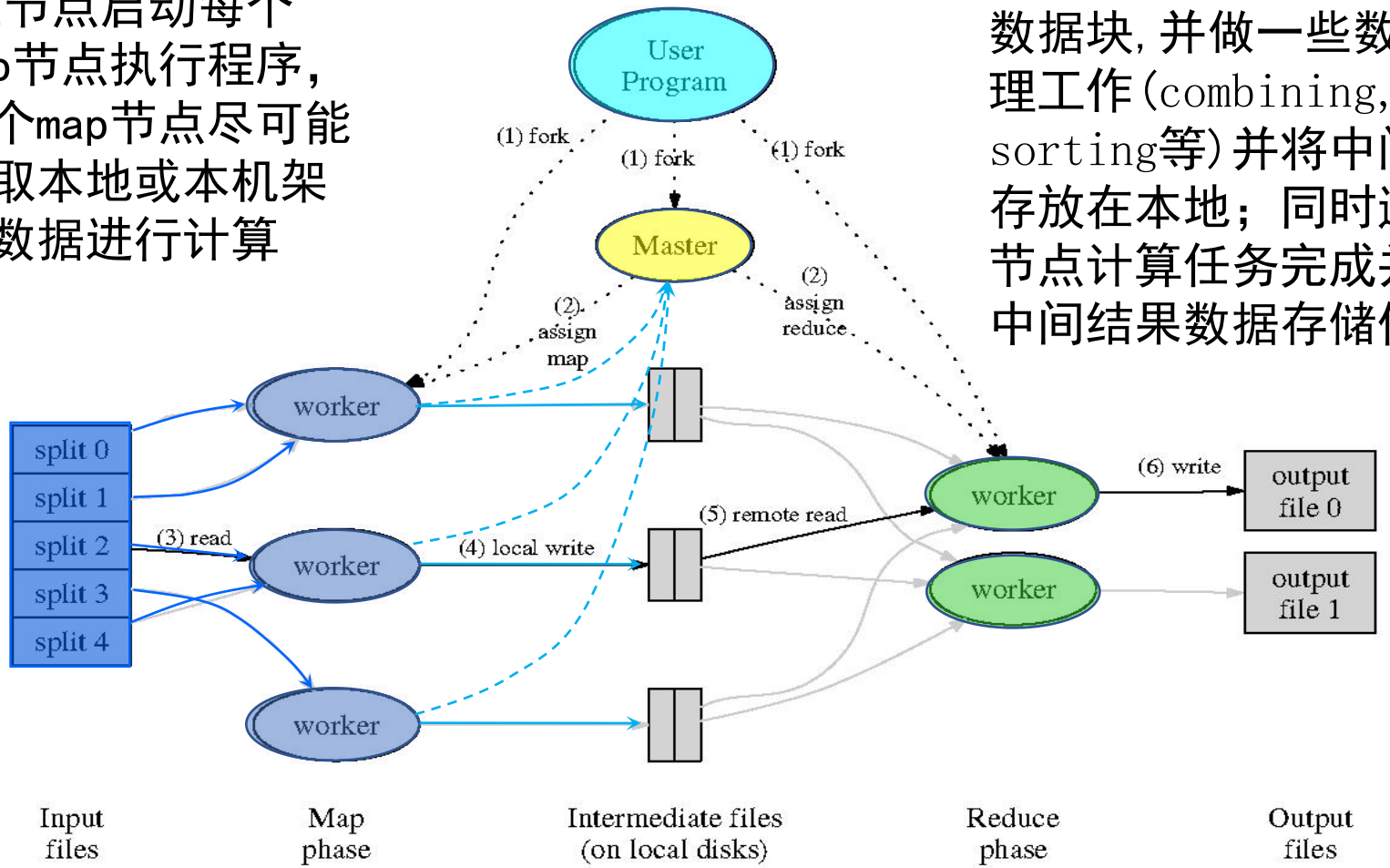
5. 主节点也为作业程序寻找和配备可用的Reduce节点，并将程序传送给Reduce节点



Cite from Dean and Ghemawat (OSDI 2004)

MapReduce并行处理的基本过程

6. 主节点启动每个Map节点执行程序，每个map节点尽可能读取本地或本机架的数据进行计算

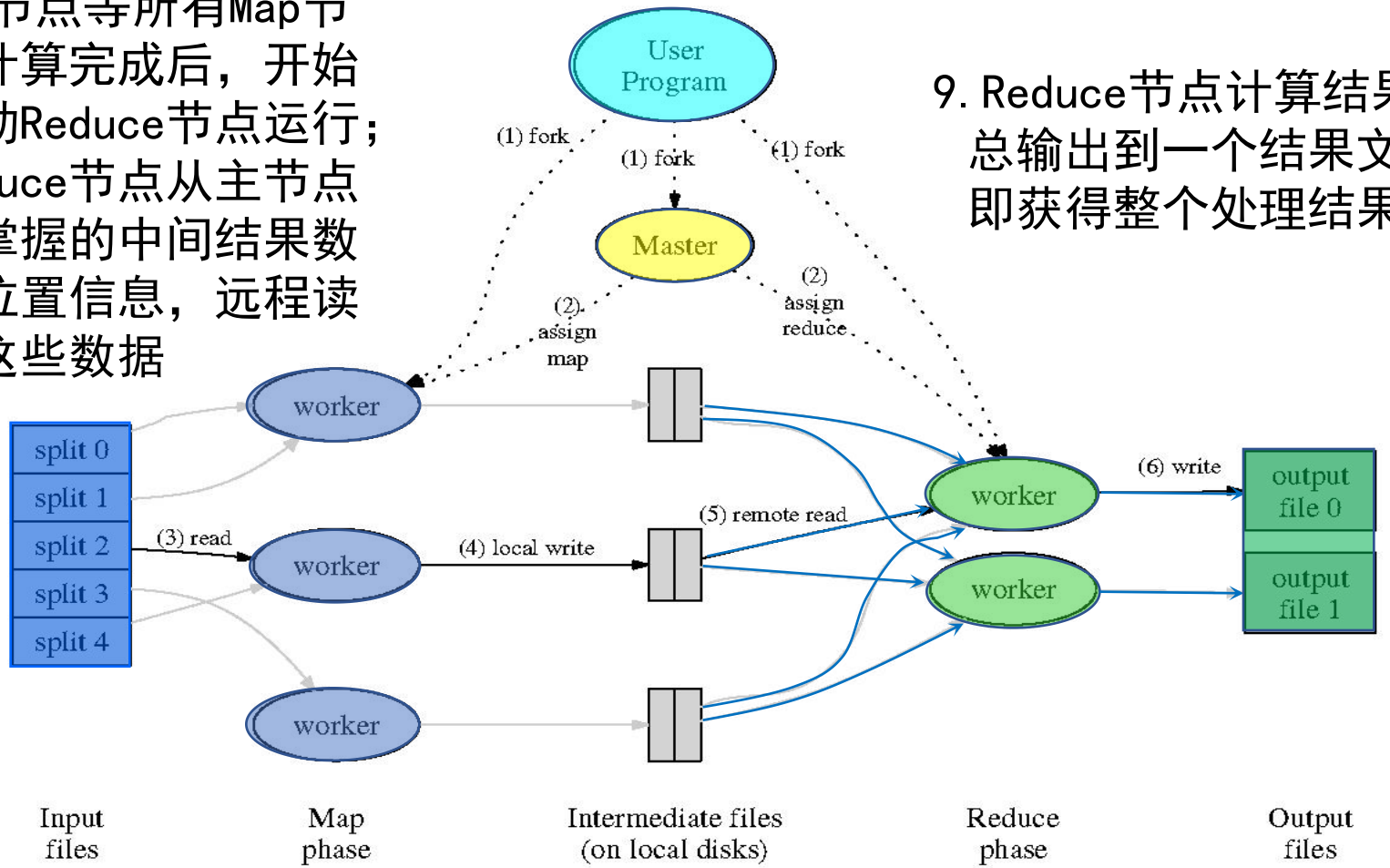


7. 每个Map节点处理读取的数据块，并做一些数据整理工作(combining, sorting等)并将中间结果存放在本地；同时通知主节点计算任务完成并告知中间结果数据存储位置

Cite from Dean and Ghemawat (OSDI 2004)

MapReduce并行处理的基本过程

8. 主节点等所有Map节点计算完成后，开始启动Reduce节点运行；Reduce节点从主节点所掌握的中间结果数据位置信息，远程读取这些数据



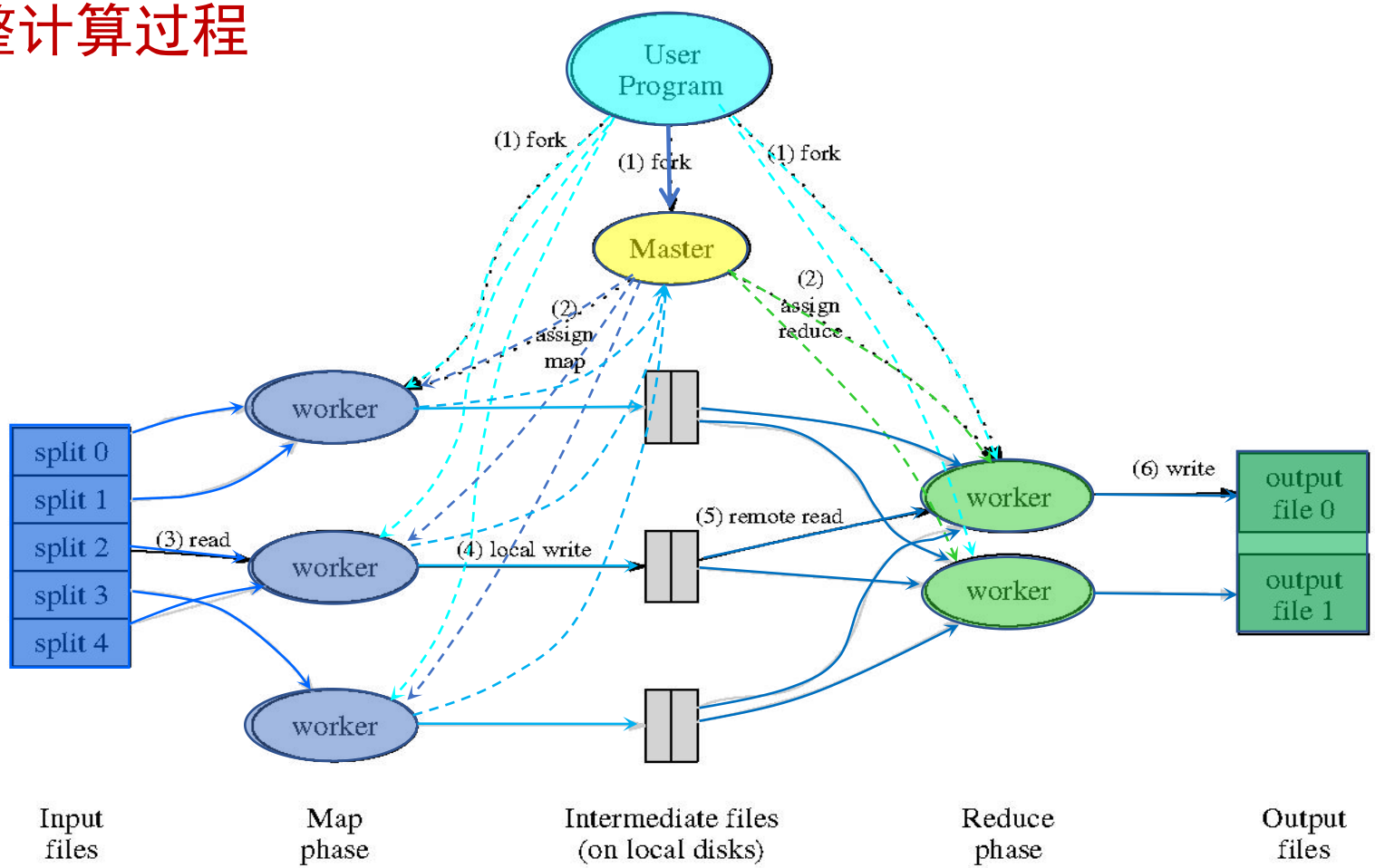
9. Reduce节点计算结果汇总输出到一个结果文件即获得整个处理结果

Cite from Dean and Ghemawat (OSDI 2004)

MapReduce并行处理的基本过程

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

完整计算过程



Cite from Dean and Ghemawat (OSDI 2004)

MapReduce大数据处理举例

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

WordCount：文档词频统计，假设有4组原始文本数据：

Text 1: the weather is good　　Text 2: today is good

Text 3: good weather is good　　Text 4: today has good weather

传统的串行处理方式(Java)：

```
String[] text = new String[]
 { "hello world", "hello every one", "say hello to everyone in the world" } ;
HashTable ht = new HashTable();
for(i=0; i<3; ++i)
{  StringTokenizer st = new StringTokenizer(text[i]);
   while (st.hasMoreTokens())
   {  String word = st.nextToken();
      if(!ht.containsKey(word)) {  ht.put(word, new Integer(1));  }
      else { int wc = ((Integer)ht.get(word)).intValue() +1;// 计数加1
             ht.put(word, new Integer(wc));
           }
   }
}
for (Iterator itr=ht.KeySet().iterator();  itr.hasNext(); )
{ String word = (String)itr.next();
  System.out.print(word+ ": "+ (Integer)ht.get(word)+";   ");
}
```

输出： good: 5;　has: 1;　is: 3;　the: 1;　today: 2;　weather: 3

WordCount：文档词频统计

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- 使用四个map节点： map: (k1; v1) → [(k2; v2)]

map节点1:
输入(k1; v1) ： (text1, “the weather is good”)
输出 [(k2; v2)]： (the, 1), (weather, 1), (is, 1), (good, 1)

map节点2:
输入： (text2, “today is good”)
输出： (today, 1), (is, 1), (good, 1)

map节点3:
输入： (text3, “good weather is good”)
输出： (good, 1), (weather, 1), (is, 1), (good, 1)

map节点4:
输入： (text3, “today has good weather”)
输出： (today, 1), (has, 1), (good, 1), (weather, 1)

- map: (k1; v1) → [(k2; v2)]

输入： 键值对(k1; v1)表示的数据

处理： 文档数据记录(如文本文件中的行，或数据表格中的行)将以“键值对”形式传入map函数；map函数将处理这些键值对，并以另一种键值对形式输出处理的一组键值对中间结果[(k2; v2)]

输出： 键值对[(k2; v2)]表示的一组中间数据

WordCount：文档词频统计

- 使用三个reduce节点： reduce: (k2; [v2]) → [(k3; v3)]

reduce节点1:

输入：(good, 1), (good, 1), (good, 1), (good, 1), (good, 1)

输出：(good, 5)

reduce节点2:

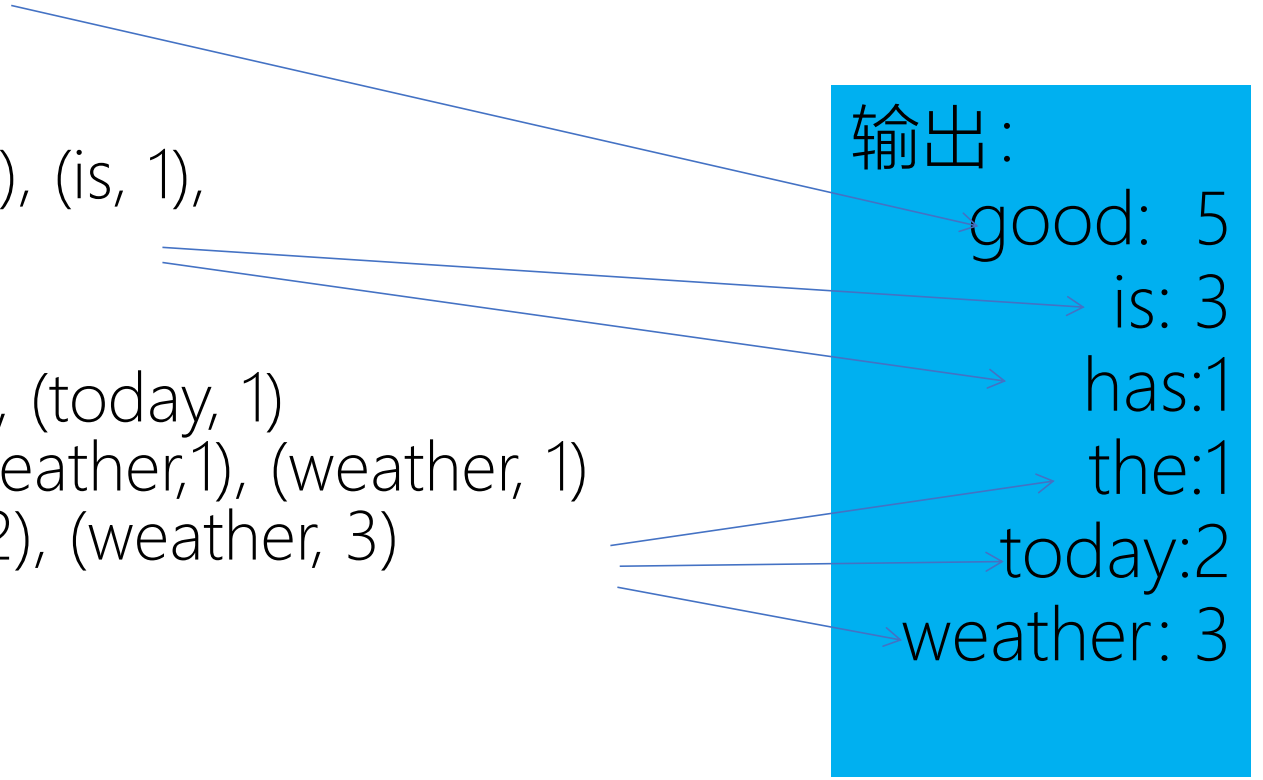
输入：(has, 1), (is,1), (is,1), (is, 1),

输出：(has, 1), (is, 3)

reduce节点3:

输入：(the, 1), (today, 1), (today, 1)
(weather, 1), (weather,1), (weather, 1)

输出：(the, 1), (today, 2), (weather, 3)



WordCount：文档词频统计

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

MapReduce伪代码（实现map和reduce两个函数）

```
Class Mapper
  method map(String input_key, String input_value):
    // input_key: text document name
    // input_value: document contents
    for each word w in input_value:
      EmitIntermediate(w, "1");
```

```
Class Reducer
  method reduce(String output_key,
                Iterator intermediate_values):
    // output_key: a word
    // output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
      result += ParseInt(v);
    Emit(AsString(result));
```

WordCount：文档词频统计-代码说明


```
package com.xyg.wordcount;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
 import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordcountMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
 @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        // 1 将maptask的文本内容先转换成String
        String line = value.toString();
        // 2 根据空格将这一行切分成单词
        String[] words = line.split(" ");
        // 3 将单词输出为<单词，1>
        for(String word:words){
        // 将单词作为key，将次数1作为value,以便于后续的数据分发，可以根据单词分发，
以便于相同单词会到相同的reducetask中
        context.write(new Text(word), new IntWritable(1));
} } }
```

WordCount：文档词频统计-代码说明


```
package com.xyg.wordcount;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
 import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordcountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
/** * key，是一组相同单词kv对的key */
@Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
        int count = 0;
        // 1 汇总各个key的个数
        for(IntWritable value:values){
        count +=value.get(); }

        // 2输出该key的总次数
        context.write(key, new IntWritable(count));
 } }
```

WordCount：文档词频统计-代码说明

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

```
package com.xyg.wordcount;
import java.io.IOException;
 import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordcountDriver {
        public static void main(String[] args) throws Exception {
        Configuration configuration = new Configuration(); // 获取配置信息
        Job job = Job.getInstance(configuration);
        job.setJarByClass(WordcountDriver.class); // 指定本程序的jar包所在的本地路径
        job.setMapperClass(WordcountMapper.class); // 指定本业务job要使用的mapper/Reducer业务类
        job.setReducerClass(WordcountReducer.class);
        job.setMapOutputKeyClass(Text.class); // 指定mapper输出数据的kv类型
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class); // 指定最终输出的数据的kv类型
        job.setOutputValueClass(IntWritable.class);
         FileInputFormat.setInputPaths(job, new Path(args[0])); // 指定job的输入原始文件所在目录
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        boolean result = job.waitForCompletion(true);
        System.exit(result?0:1);
} }
```

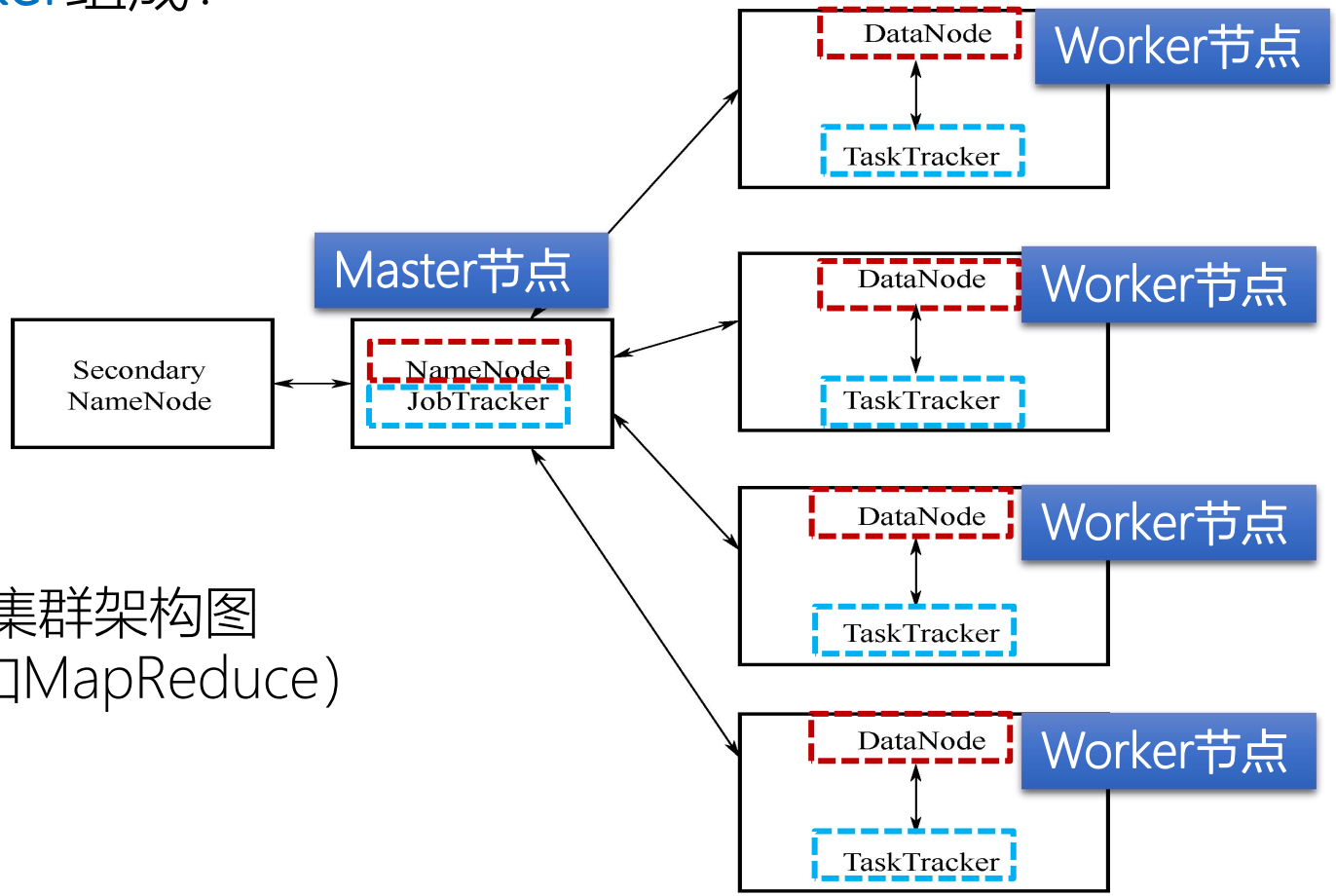
大数据处理—MapReduce框架

北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

MapReduce的运行原理

Hadoop集群架构

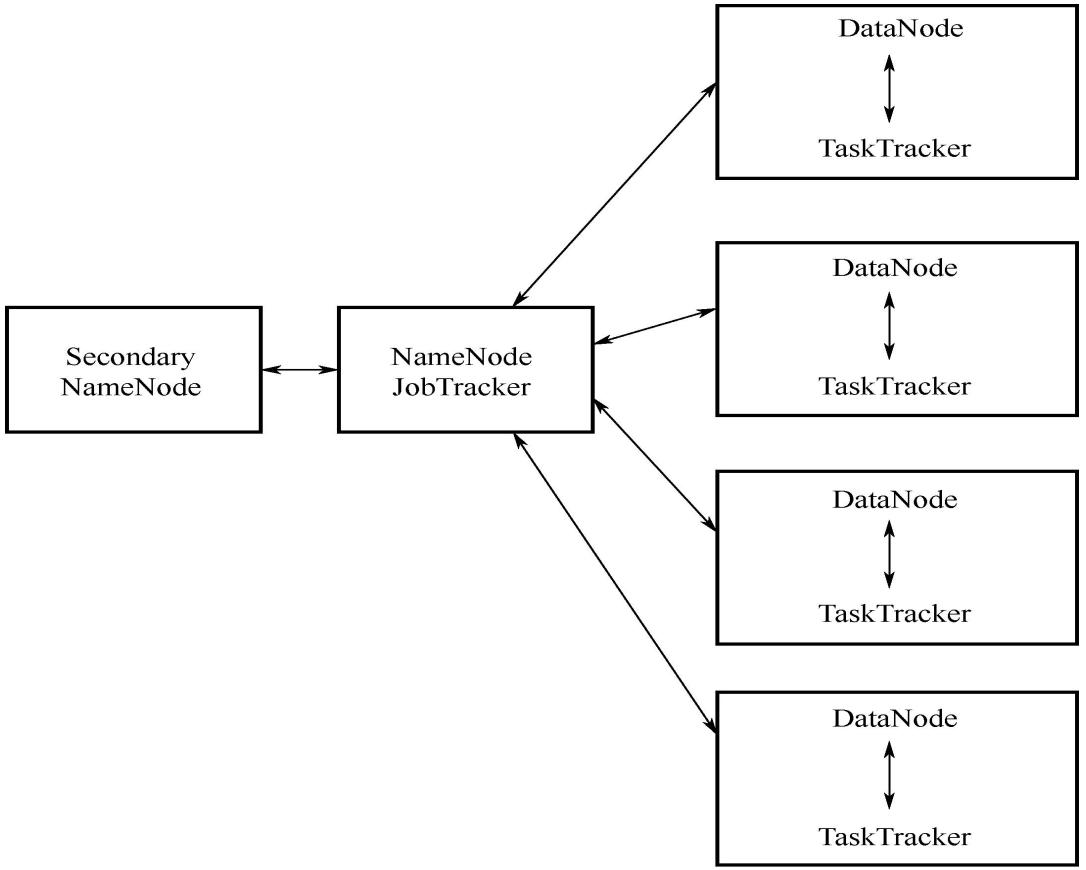
在分布式存储和计算方面，Hadoop采用的是主/从(Master/Slave)架构，集群主要由 NameNode，DataNode，Secondary NameNode，JobTracker，TaskTracker组成：



Hadoop集群架构图
(包括HDFS和MapReduce)

JobTracker

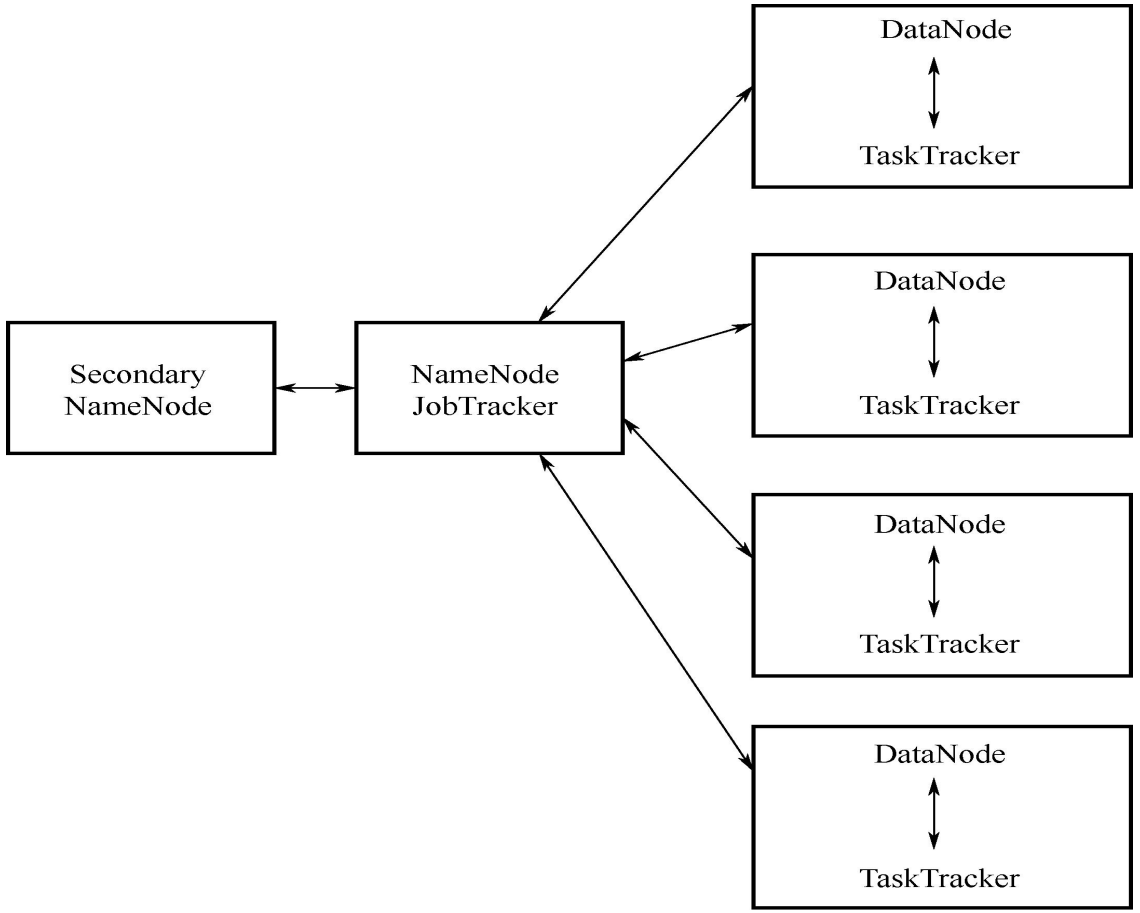
- JobTracker主要负责资源监控和作业调度
- JobTracker监控所有TaskTracker与作业的健康状况，一旦发现失败情况后，其会将相应的任务转移到其他节点
- JobTracker会跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器，而调度器会在资源出现空闲时，选择合适的任务使用这些资源



TaskTracker

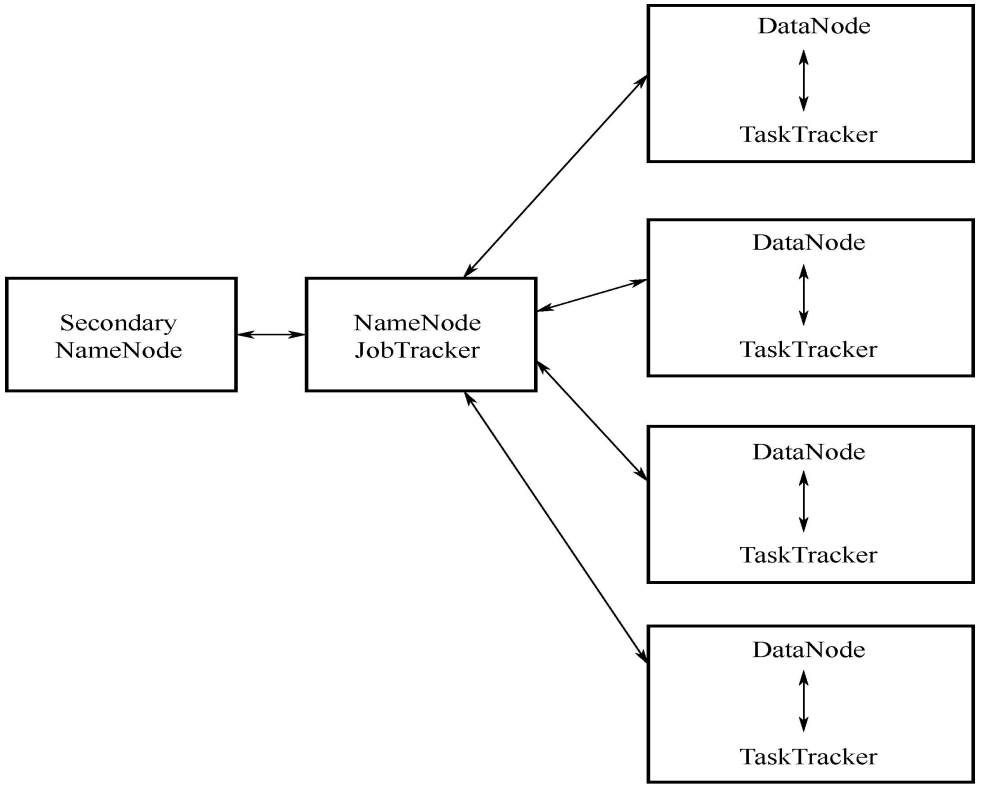
北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- TaskTracker负责作业的执行
- 周期性地通过Heartbeat将本节点上资源的使用情况和任务的运行进度汇报给JobTracker
- 接收JobTracker发送过来的命令并执行相应的操作（如启动新任务、杀死任务）



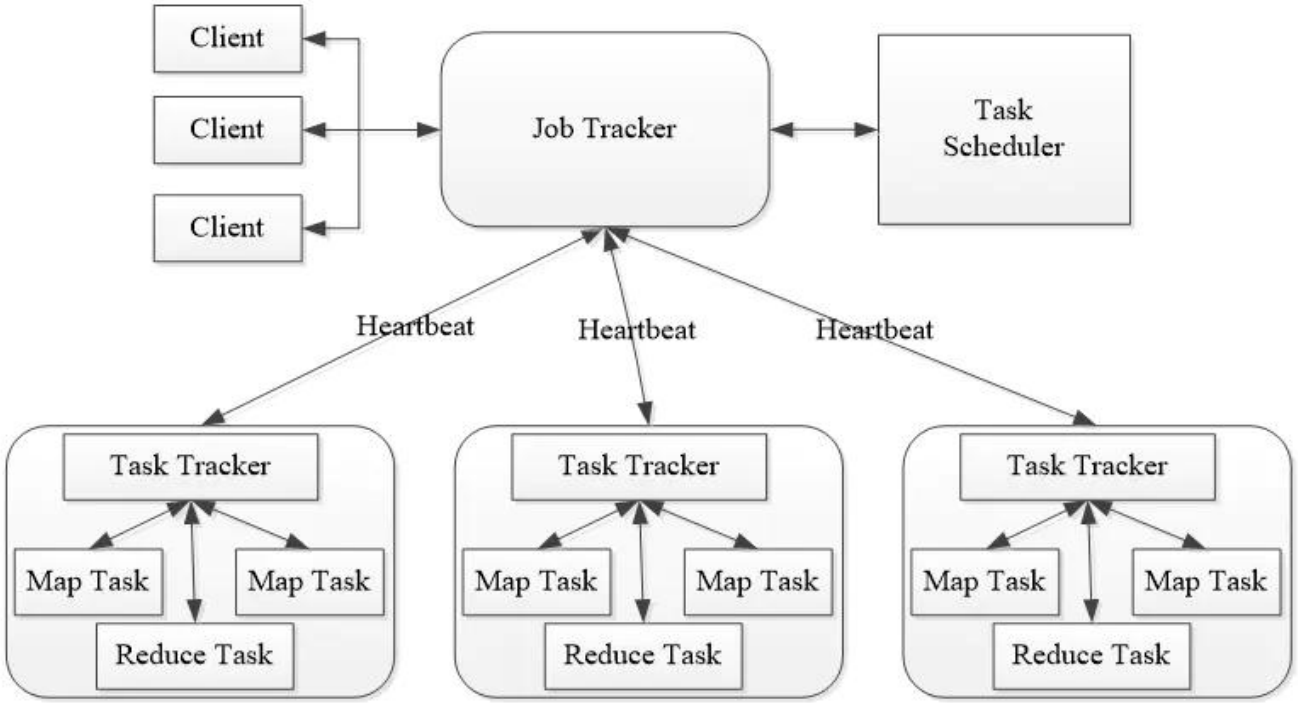
MapReduce运行原理

- TaskTracker使用“slot”等量划分本节点上的资源量
- slot是Hadoop的资源单位代表计算资源（CPU、内存等）
- slot是一个逻辑概念，一个节点的slot的数量用来表示某个节点的资源的容量或者说是能力的大小
- 一个Task获取到一个slot后才有机会运行，而Hadoop调度器的作用就是将各个TaskTracker上的空闲slot分配给Task使用



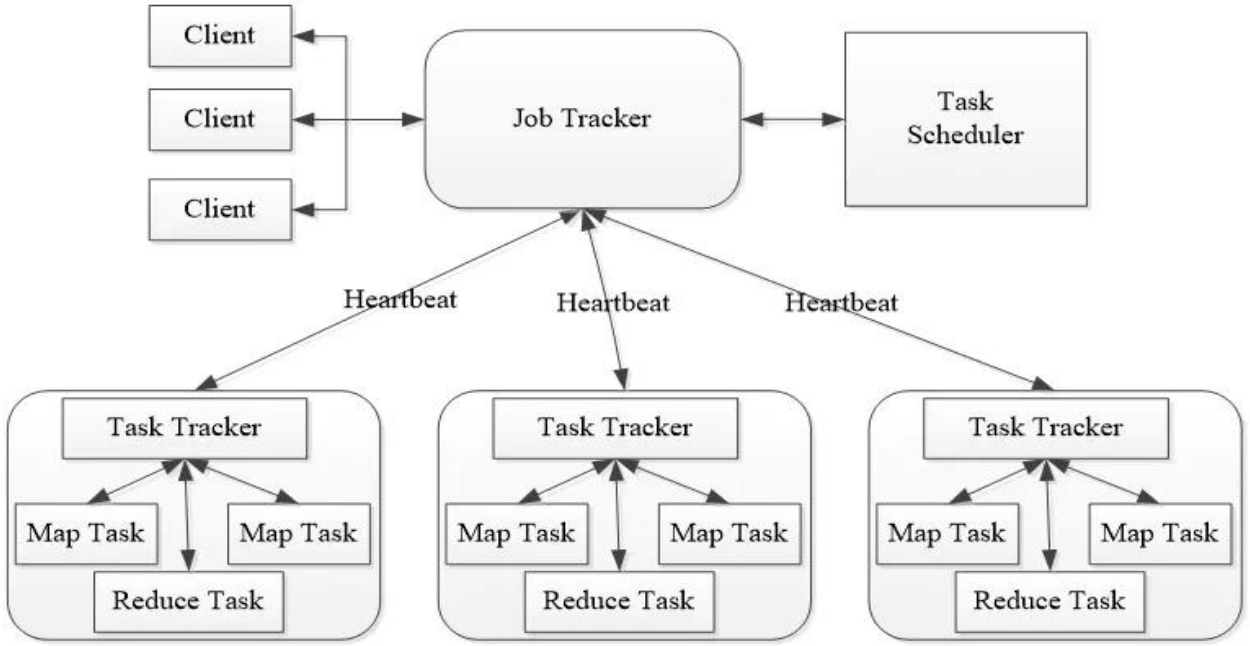
MapReduce运行原理

- 一个MapReduce程序可对应若干个作业，而每个作业会被分解成若干个Map/Reduce任务（Task）
- Task分为Map Task和Reduce Task两种，slot分为Map slot和Reduce slot两种，分别供Map Task和Reduce Task使用
- TaskTracker通过slot数目（可配置参数）限定Task的并发度



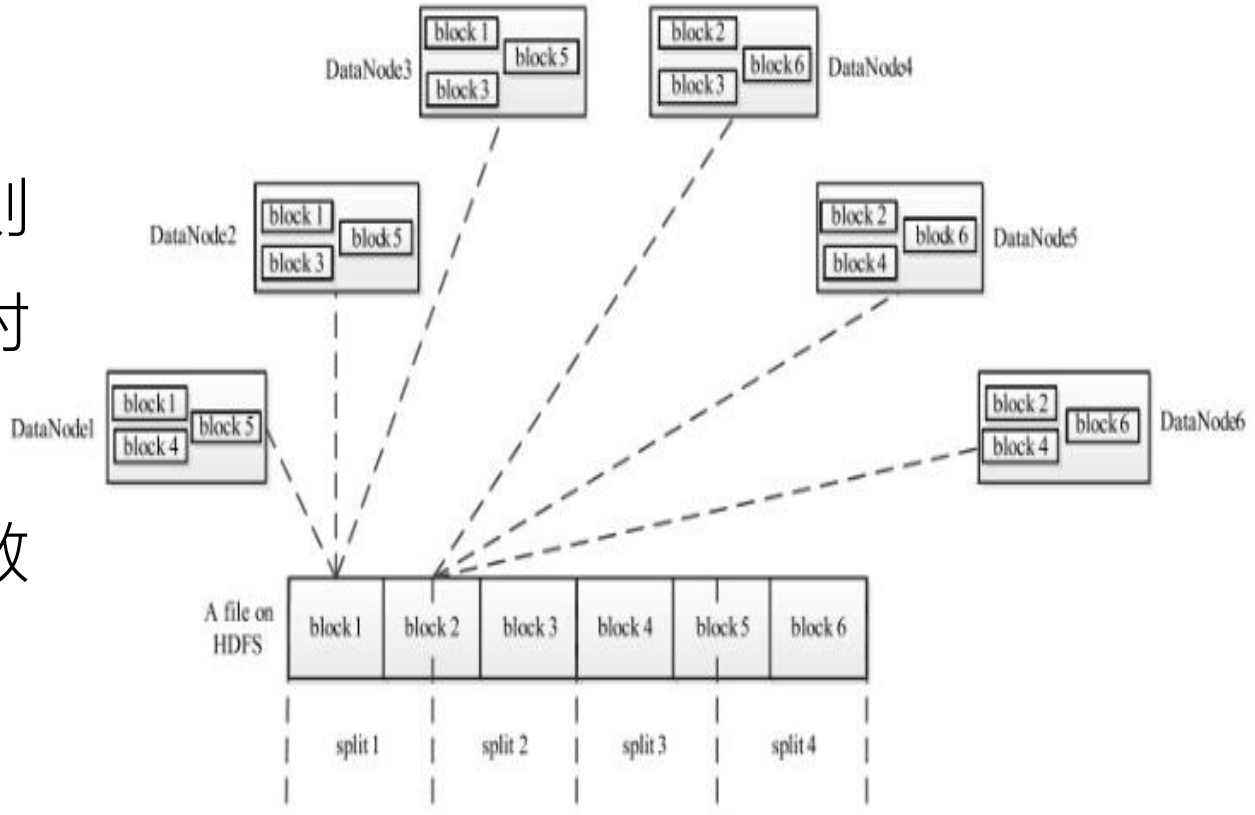
MapReduce详细架构

- Client：用户可以编写MapReduce程序，通过Client提交到JobTracker端
- 每一个 Job 都会在用户端，将应用程序以及配置参数 Configuration 打包成 JAR 文件
- 将JAR 文件存储在HDFS，并把路径提交到 JobTracker 的 master 服务
- 由 master 创建每一个Task（即 MapTask 和 ReduceTask） 将它们分发到各个 TaskTracker 服务中去执行



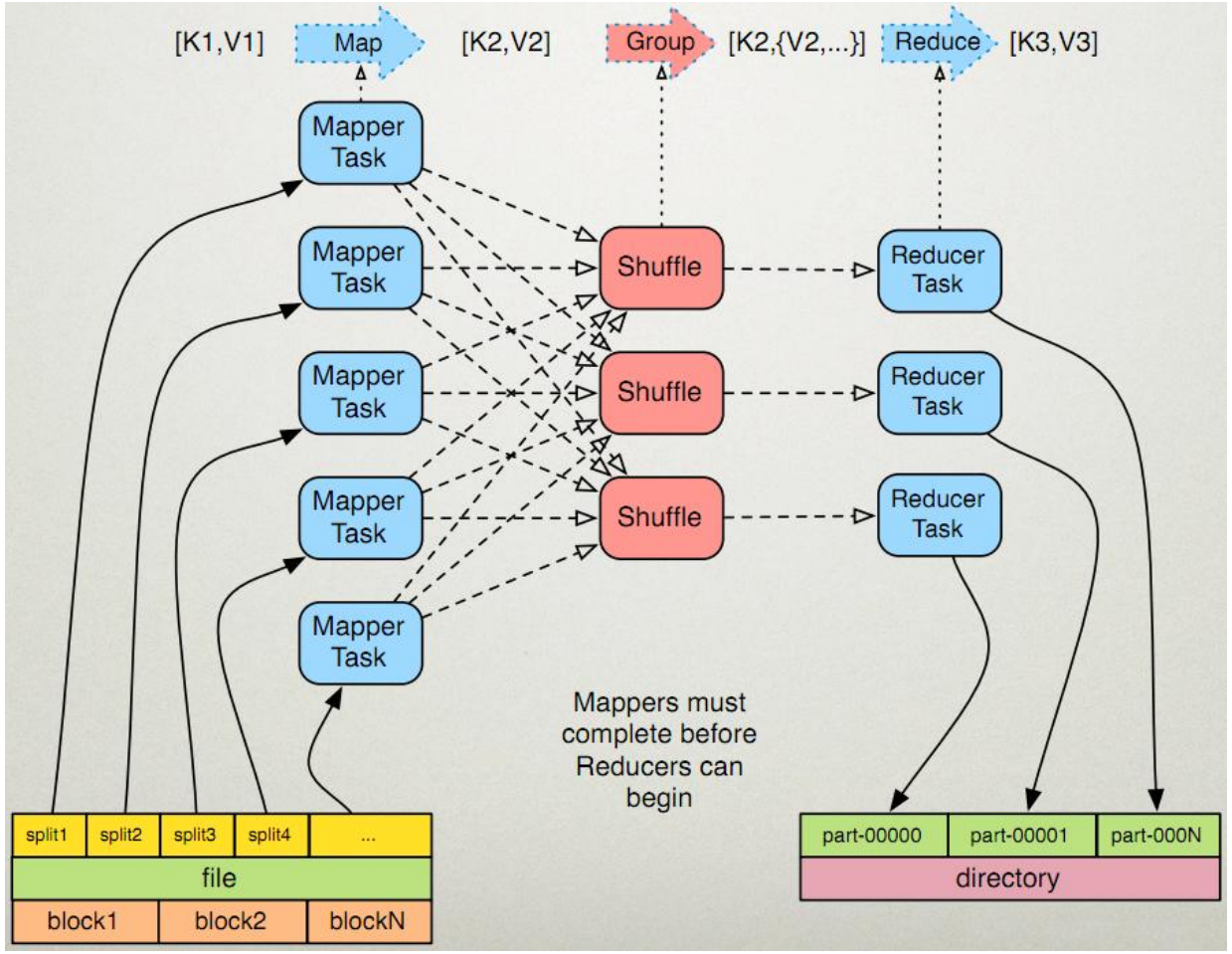
Map阶段数据处理单位split

- HDFS以block为基本单位存储数据，MapReduce的Map阶段处理单位是split
- split分片：将数据源根据用户自定义规则分成若干个分片数据集；一个split分片对应一个Map Task
- split是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在节点等；划分方法由用户决定



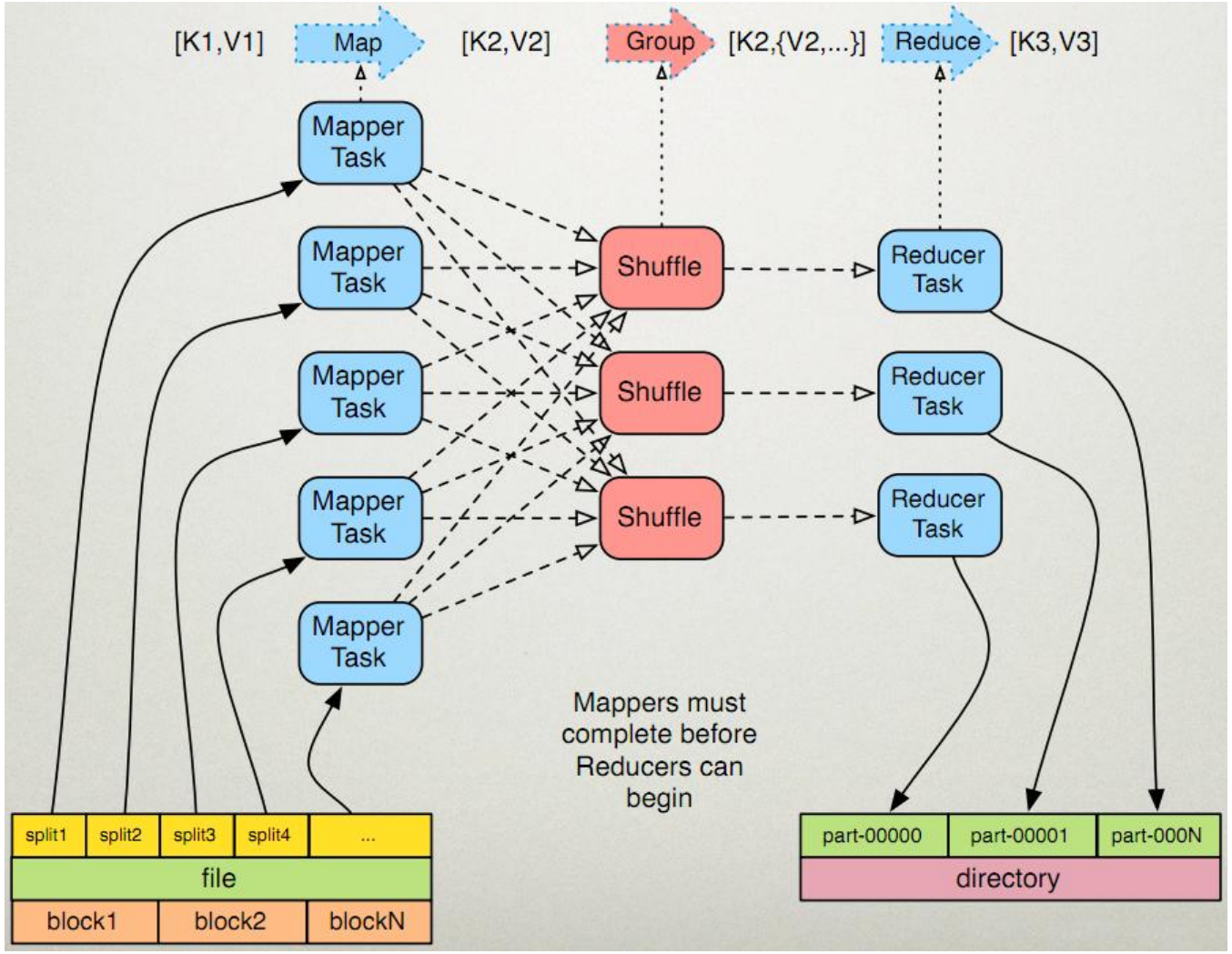
Map阶段数据处理单位split

- 输入分片不是物理分片，而是逻辑分片：不是把原来的一个大文件，如10MB的文件，切分成10个1MB的小文件
- 逻辑分片就是根据文件的字节索引进行分割，比如0~1MB位置定义为第一个分片，1MB~2MB定义为为第二个分片，依次类推.....
- 输入分片（input split）存储的并非数据本身，而是一个分片长度和一个记录数据的位置的数组



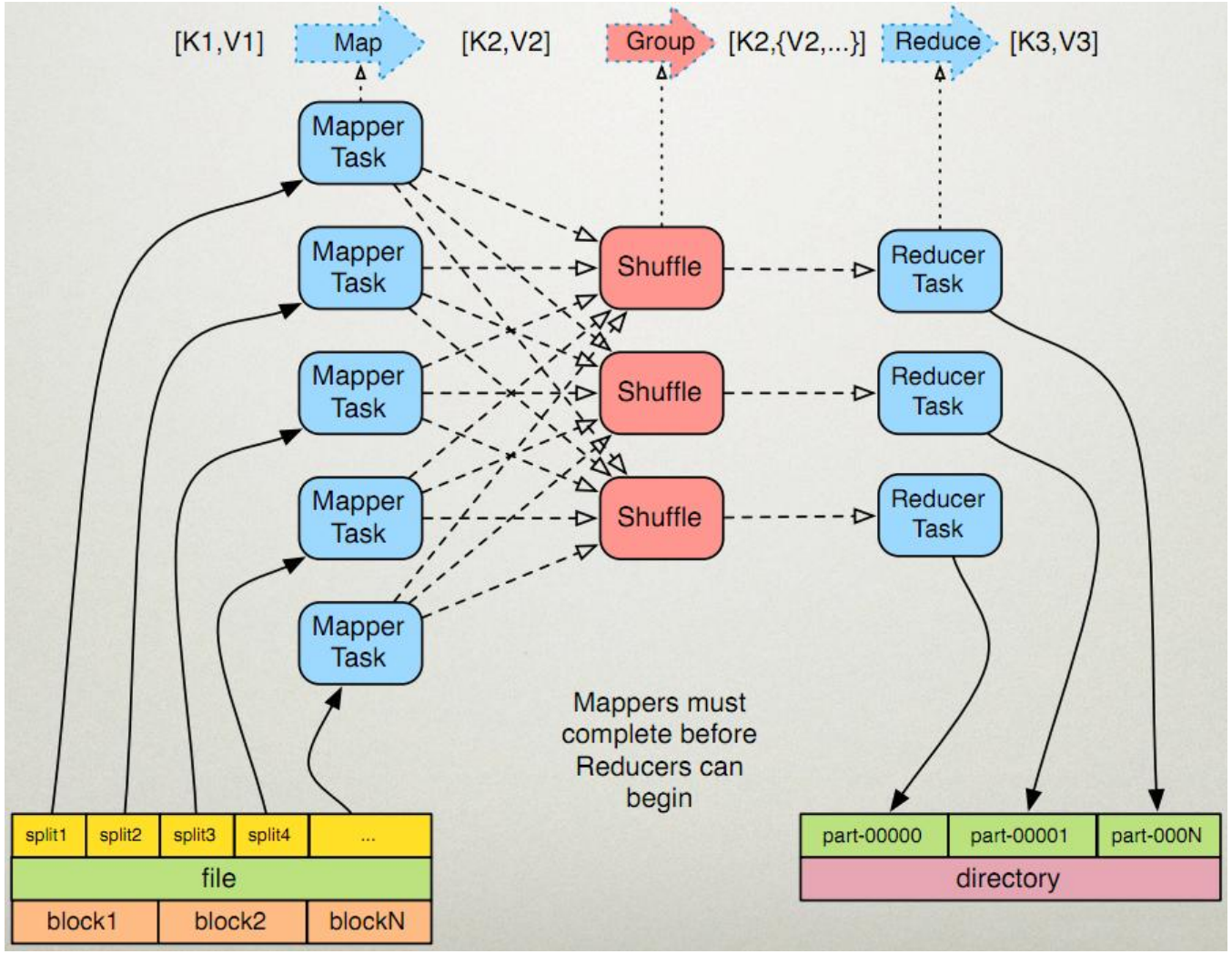
Map阶段数据处理单位split

- Map Task的个数等于split的个数
  - mapreduce在处理大文件的时候，会根据一定的规则，把大文件划分成多个分片，提高map的并行度
  - 划分出来的就是InputSplit，每个map处理一个InputSplit
  - 多少个InputSplit，对应多少个Map task



Reduce阶段的数据处理单位Partition

- Partition分区：将整个大数据块分成多个数据块
- 一个Partition分区对应一个Reduce Task
- Reduce 的个数决定了输出文件的个数



MapReduce执行-第一阶段split分片

- 分片主要是InputFormat类来负责划分Split
- FileInputFormat是InputFormat的子类，是使用比较广泛的类
- 输入格式如果是HDFS上的文件，基本上用的都是FileInputFormat的子类，如：
  - TextInputFormat处理普通的文件，SequceFileInputFormat处理Sequce格式文件
  - FileInputFormat类中的getSplits(JobContext job)方法是划分split的主要逻辑
- 每个输入分片的大小是固定的，默认情况下，输入片(InputSplit)的大小与数据块(Block)的大小是相同的
  - Hadoop 2.x默认的block大小是128MB，Hadoop 1.x默认的block大小是64MB，可以在hdfs-site.xml中设置dfs.block.size，注意单位是byte
  - 分片大小范围可以在mapred-site.xml中设置

MapReduce执行-第一阶段split分片

北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

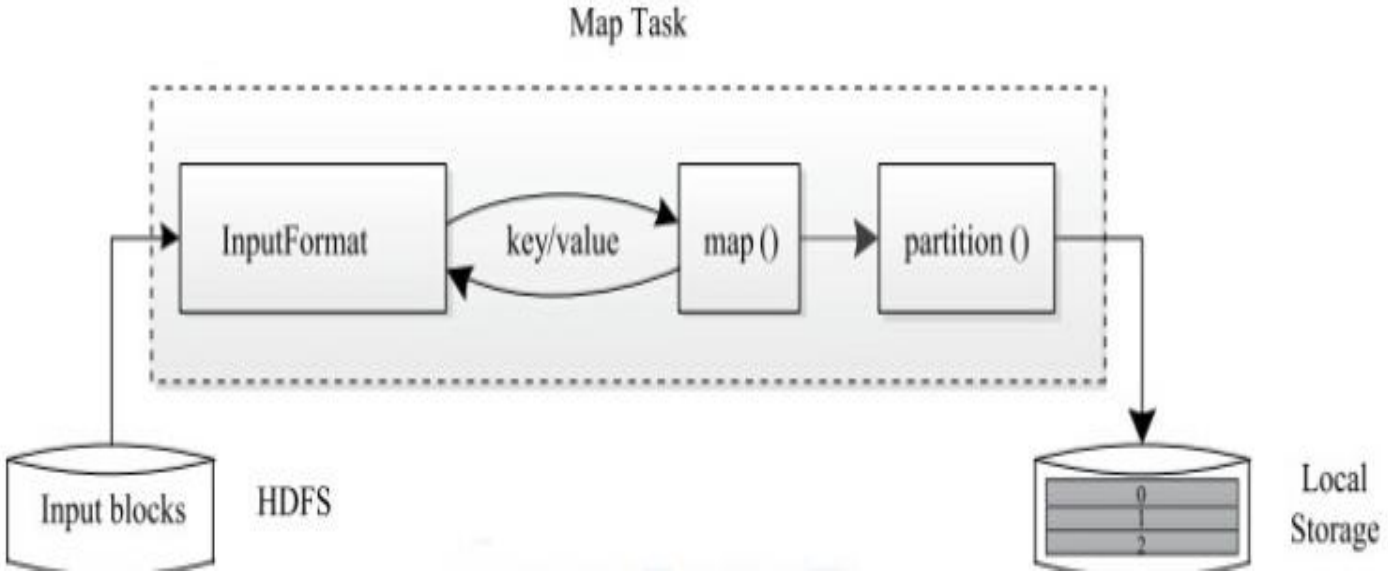
默认分片大小与Block分块大小是相同的原因?

- ✓ hadoop在存储有输入数据（HDFS中的数据）的节点上运行map任务，可以获得高性能，这就是所谓的数据本地化。
- ✓ 所以分片大小默认与HDFS上的块大小一样，减少网络传输数据，使用本地数据运行map任务。
- ✓ 如果分片跨越2个数据块，对于任何一个HDFS节点（Hadoop系统保证一个块存储在一个datanode上，基本不可能同时存储这2个数据块），分片中的另外一块数据就需要通过网络传输到map任务节点，与使用本地数据运行map任务相比，效率则更低。

MapReduce执行-第二阶段Map阶段

- 每个Mapper任务是一个Java进程，它会读取HDFS文件中自己对应的输入切片
- Map Task先将对应的split迭代解析成一个个key/value对
- 依次调用用户自定义的map()函数进行处理
- 将切片中记录按照一定的规则解析成很多的键值对

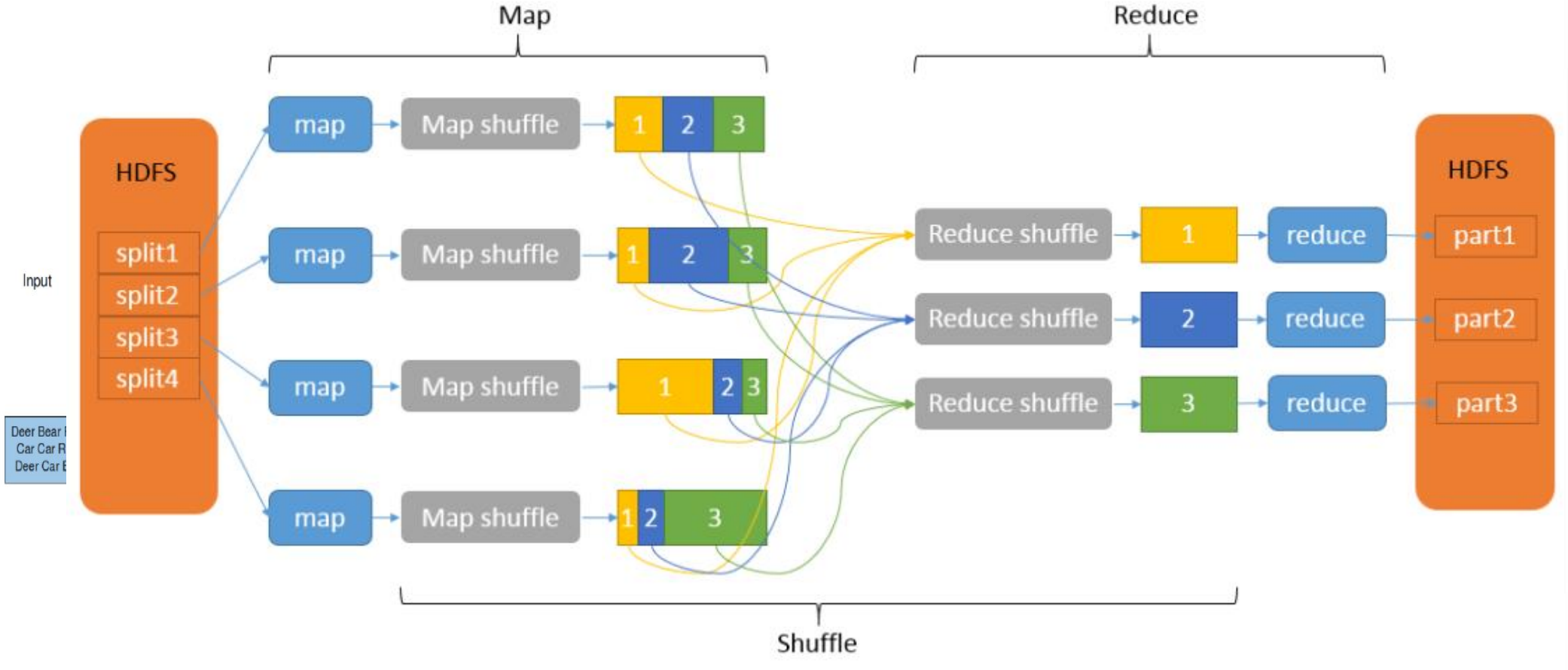
- InputFormat类2个重要作用：
  - 将输入的数据切分为多个逻辑上的InputSplit，其中每一个InputSplit作为一个map的输入
  - 提供一个RecordReader，用于将InputSplit的内容转换为可以作为map输入的k,v键值对



Map Task将临时结果存放到本地磁盘上，其中临时数据被分成若干个partition

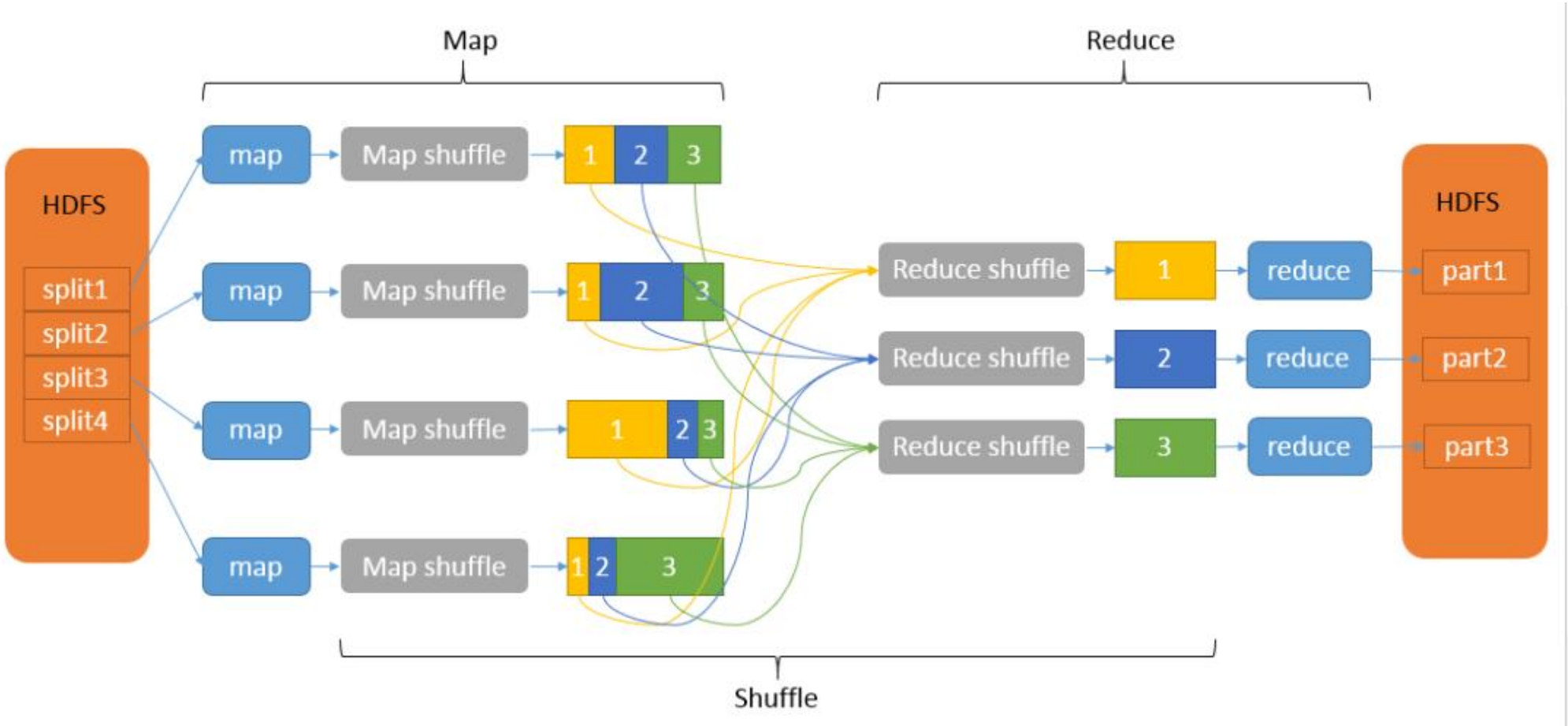
MapReduce执行-第三阶段Shuffle阶段

- Shuffle阶段也称为洗牌阶段，该阶段是将输出的<k2,v2>传给Shuffle（洗牌）,Shuffle完成对数据的分区、排序和合并等操作
- Shuffle过程包含在Map和Reduce两端，即Map shuffle和Reduce shuffle，Shuffle描述着数据从map task流向reduce task的这段过程



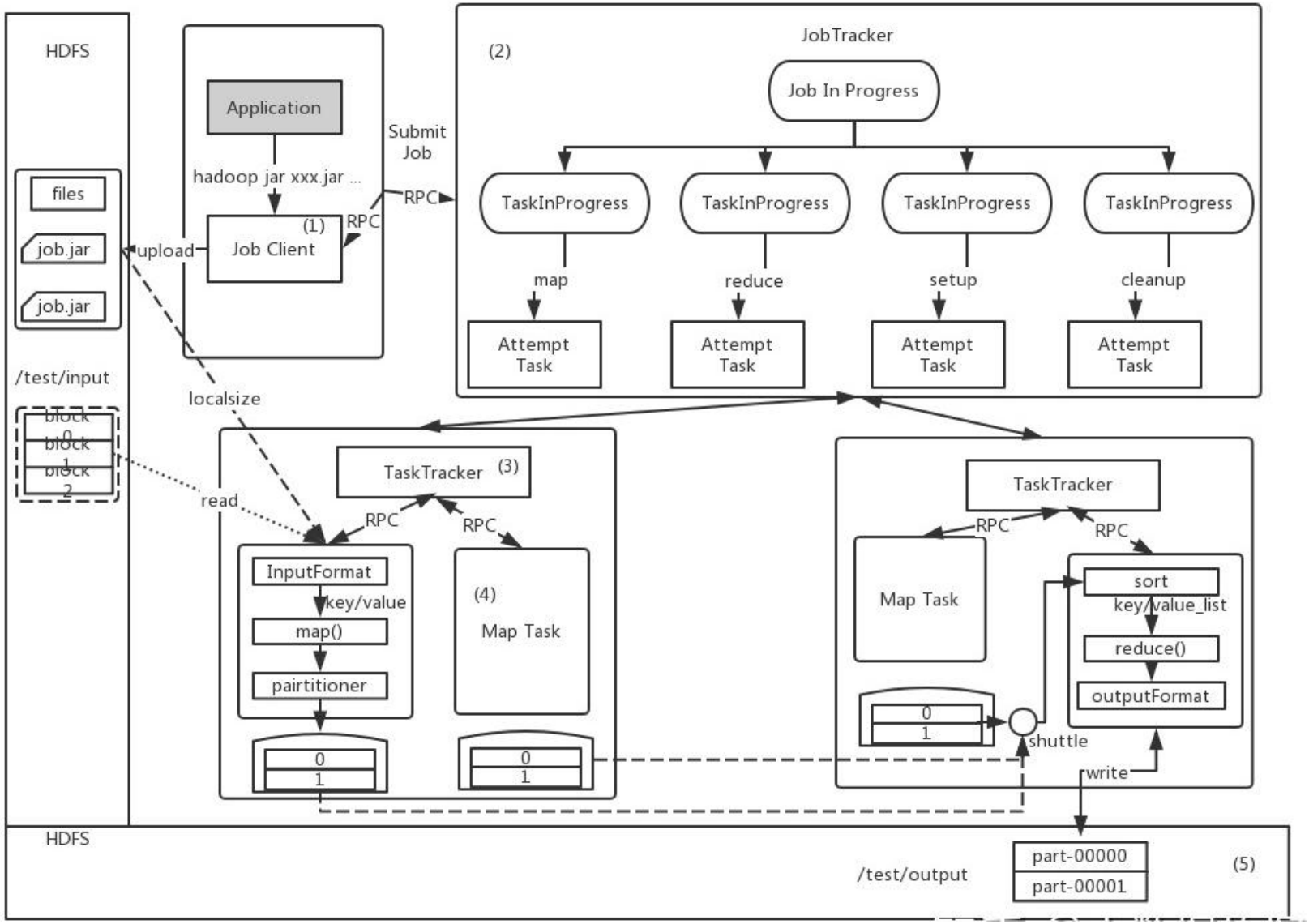
MapReduce执行-第四阶段Reduce阶段

- 依次读取<key, value list>，调用用户自定义的reduce()函数处理，并将最终结果存到HDFS上（称为“Reduce阶段”）



完整生命周期

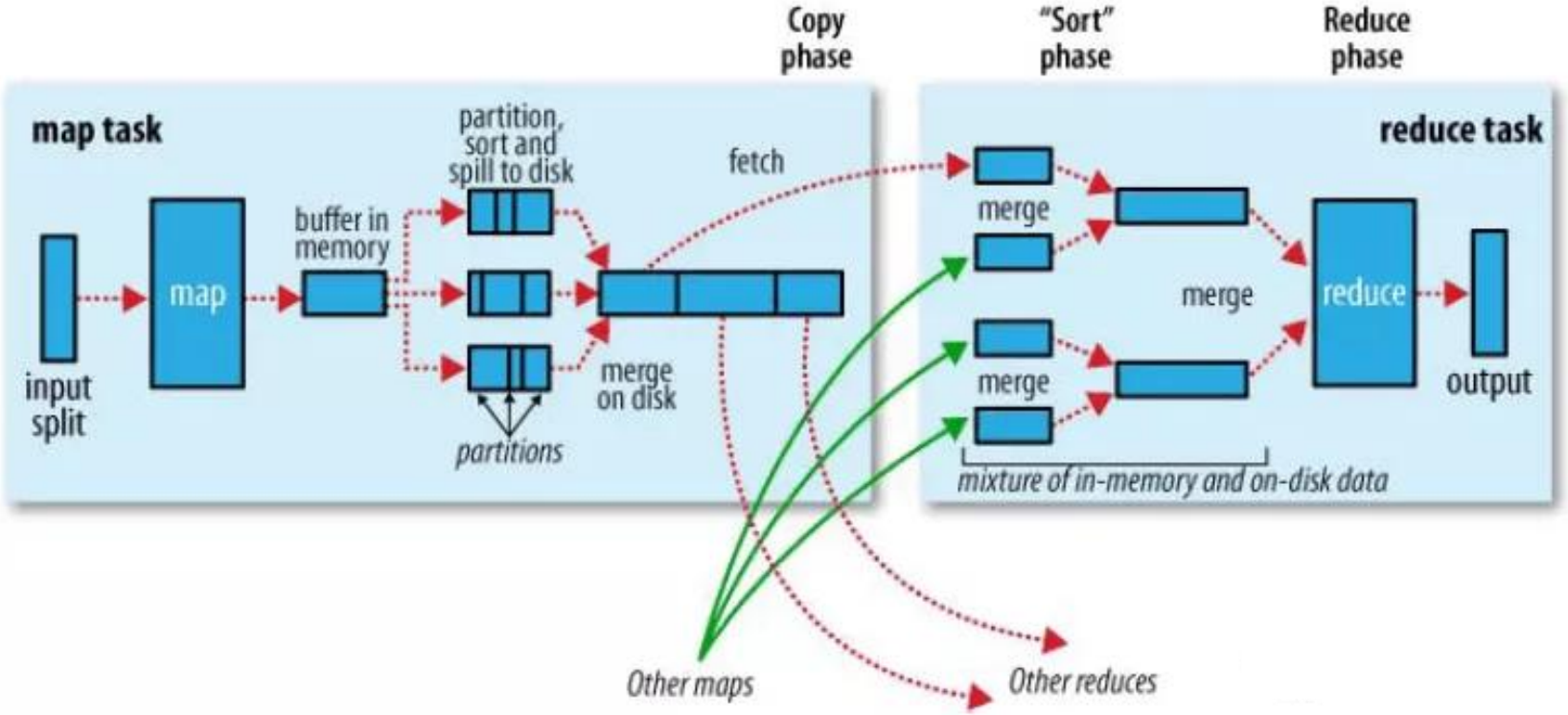
- ✓ 当用户 (Application)提交作业后，JobClient 将作业所需要的相关文件上传到HDFS，并通过RPC（远程调用协议）通知 JobTracker
- ✓ JobTracker 内的任务调度模块为作业创建一个JobInProgress对象，用来跟踪作业的实时运行状况
- ✓ JobInProgress 为每个Task创建一个TaskInProgress 对象，用来跟踪子任务的运行状态。
- ✓ TaskTracker为任务运行准备环境，每个TaskTracker可以运行多个Task，TaskTracher会为每个Task启动一个独立的JVM以避免Task之间的相互影响
- ✓ 环境准备完毕后，TaskTracker便会启动Task。
- ✓ 所有的Task执行完毕，MapReudce作业生命周期结束。



Map端的Shuffle 与 Reduce端的 Shuffle 的补充

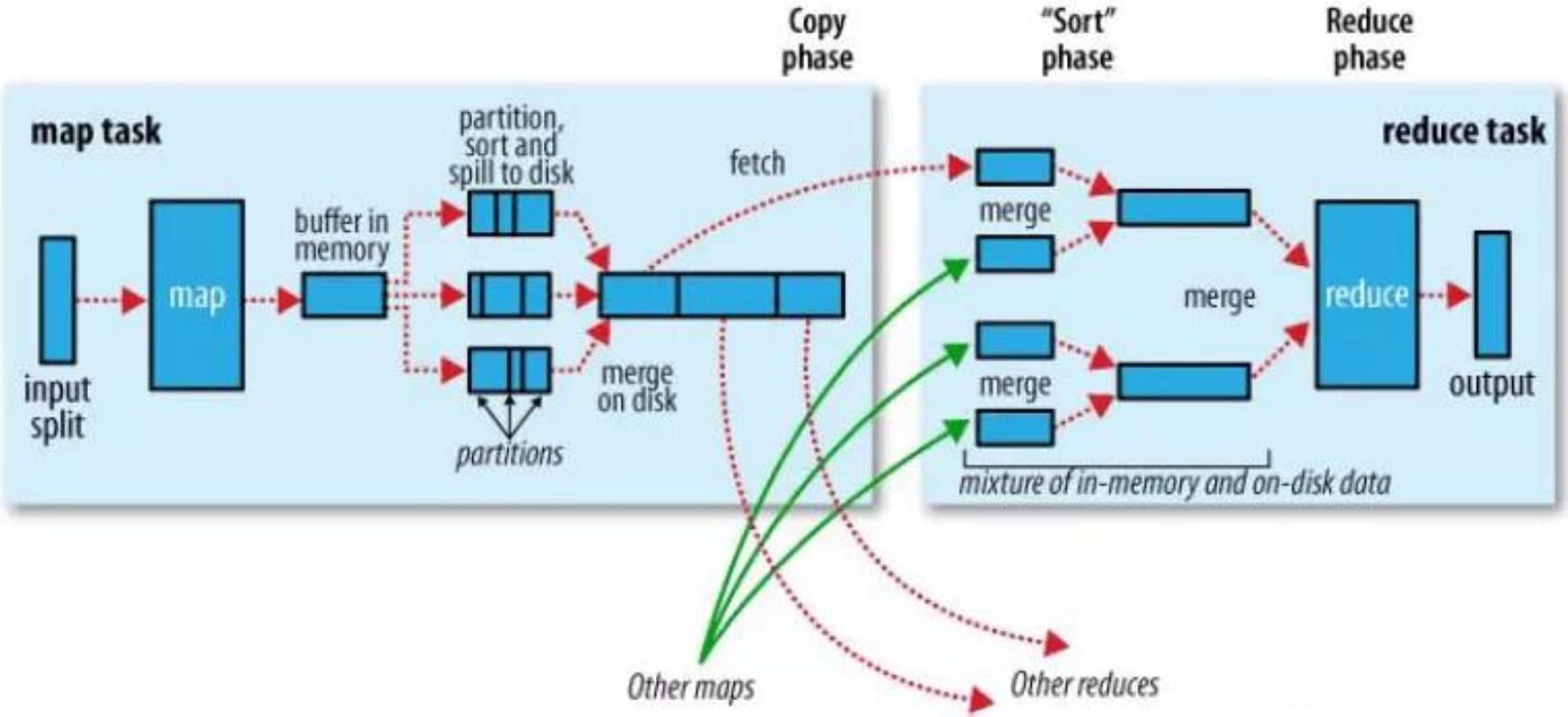
Map端的Shuffle

- 每个输入分片会让一个 Map 任务来处理，默认情况下，以 HDFS 一个块的大小为一个分片
- Map 函数开始产生输出时，并不是简单地把数据写到磁盘，频繁的磁盘操作会导致性能严重下降
- 首先把数据写到内存中的一个缓冲区，并做一些预排序，以提升效率
- 每个 Map 任务都有一个用来写入输出数据的循环内存缓冲区（默认大小为 128MB）
- 当缓冲区中的数据量达到一个特定阈值（默认是 80%）时，系统将会启动一个后台线程，把缓冲区中的内容写到磁盘中（即 Spill 阶段）



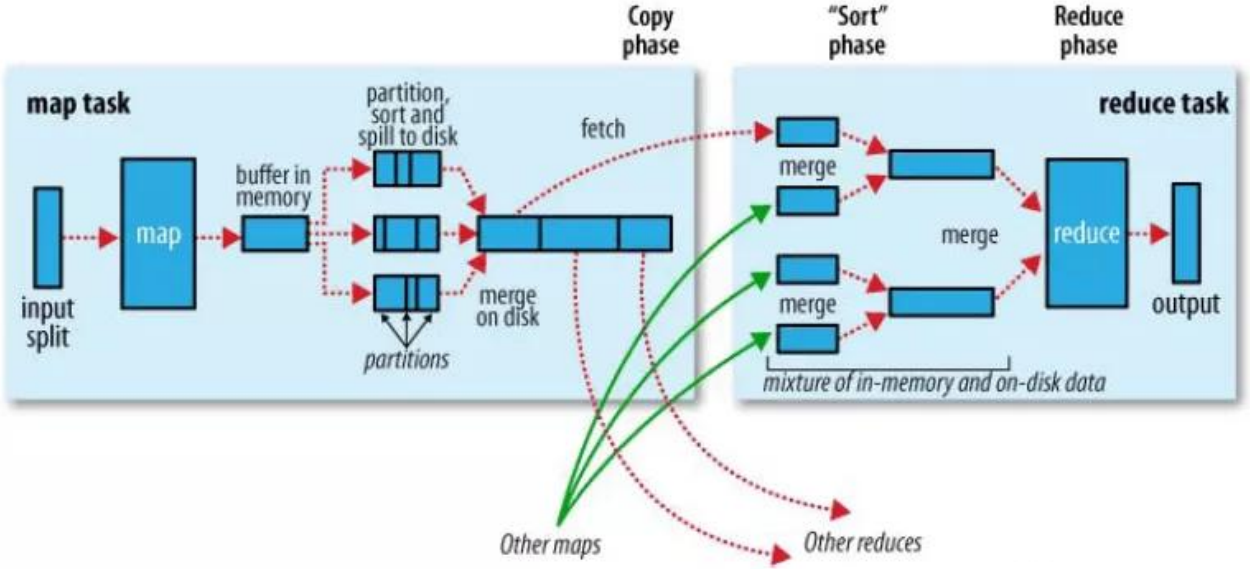
Map端的Shuffle

- 在写磁盘过程中，Map 输出继续被写到缓冲区中，但如果在此期间缓冲区被填满，那么 Map 任务就会阻塞直到写磁盘过程完成
- 在写入磁盘之前，线程首先根据reduce任务的数目将数据划分为相同数目的分区（Partition），也就是一个reduce任务对应一个分区的数据
- 这样做是为了避免有些reduce任务分配到大量数据，而有些reduce任务却分到很少数据，甚至没有分到数据的尴尬局面



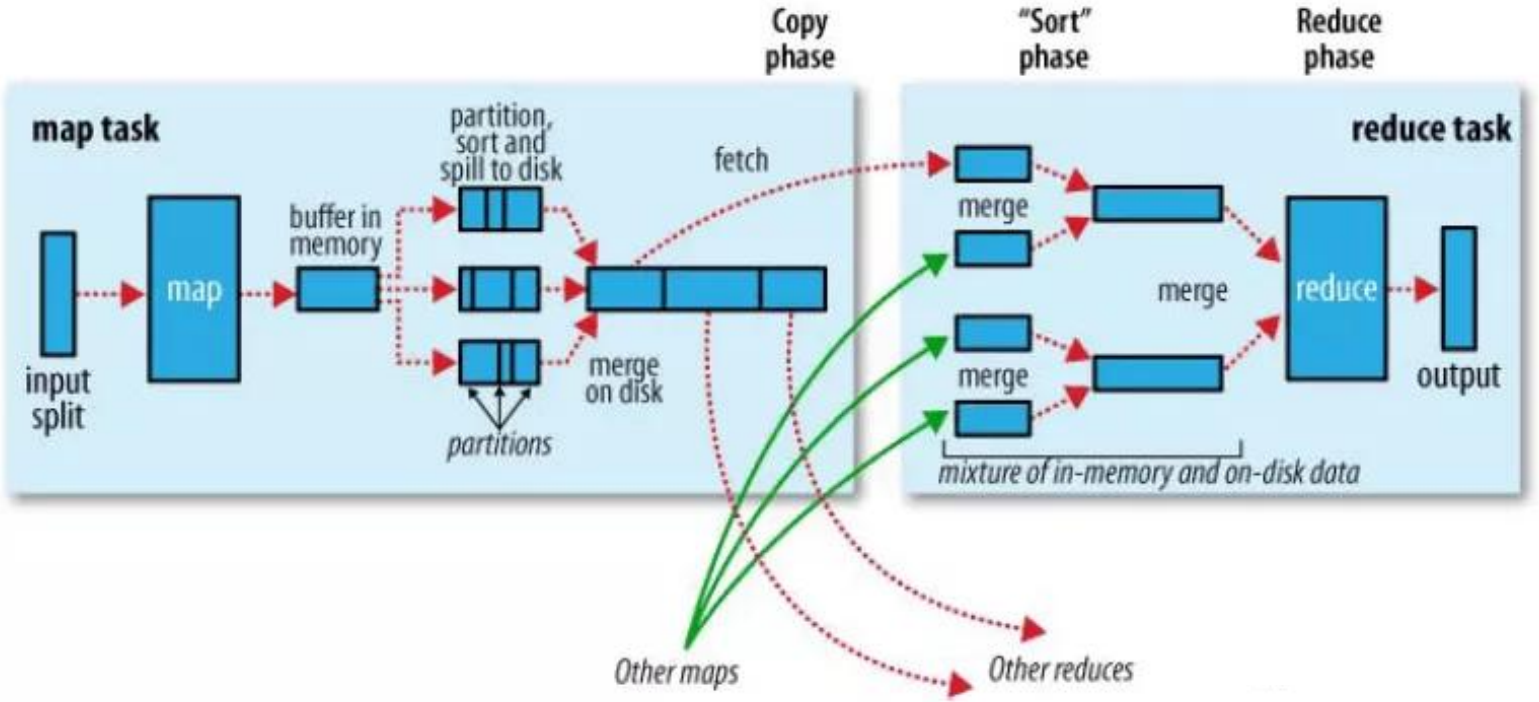
Map端的Shuffle

- **MapTask对数据以分区为单位进行合并。**对于每个分区采用多轮递归合并的方式
- 合并的过程中会不断地进行排序和Combine操作，目的有两个：
  - ①尽量减少每次写入磁盘的数据量
  - ②尽量减少下一复制阶段网络传输的数据量
  - 最后合并成了一个已分区且已排序的文件
- 让每个MapTask最终只生成一个数据文件，避免同时打开大量文件和同时读取大量小文件产生的随机读取带来的开销
- 合并完之后数据（以key和value的形式存在，及<k2,v2>）会基于Partition被发送到不同的Reduce上，Reduce会从不同的Map上取得属于自己的数据并写入磁盘，完成merge操作减少文件数量，并调用Reduce程序，最终Output完成输出
- 归并完毕后Map 任务告知 TaskTracker 任务已完成
- 只要其中一个 Map 任务完成，Reduce 任务就会开始复制它的输出（Copy 阶段）
- Map 任务的输出文件放置在运行 Map 任务的 TaskTracker 的本地磁盘上，它是运行 Reduce 任务的 TaskTracker 所需要的输入数据



Reduce 端的 Shuffle 阶段

- Reduce 进程启动数据复制线程，请求 Map 任务所在的 TaskTracker 以获取输出文件,并且每个map传来的数据都是有序的（Copy 阶段）
- 如果reduce端接受的数据量相当小，则直接存储在内存中，如果数据量超过了该缓冲区大小的一定比例，则对数据合并后溢写到磁盘中
- 随着溢写文件的增多，后台线程会将它们合并成一个更大的有序的文件，给后面的合并节省时间
- 在map端和reduce端，MapReduce都是反复执行排序、合并操作：排序是hadoop的灵魂



大数据处理—MapReduce框架

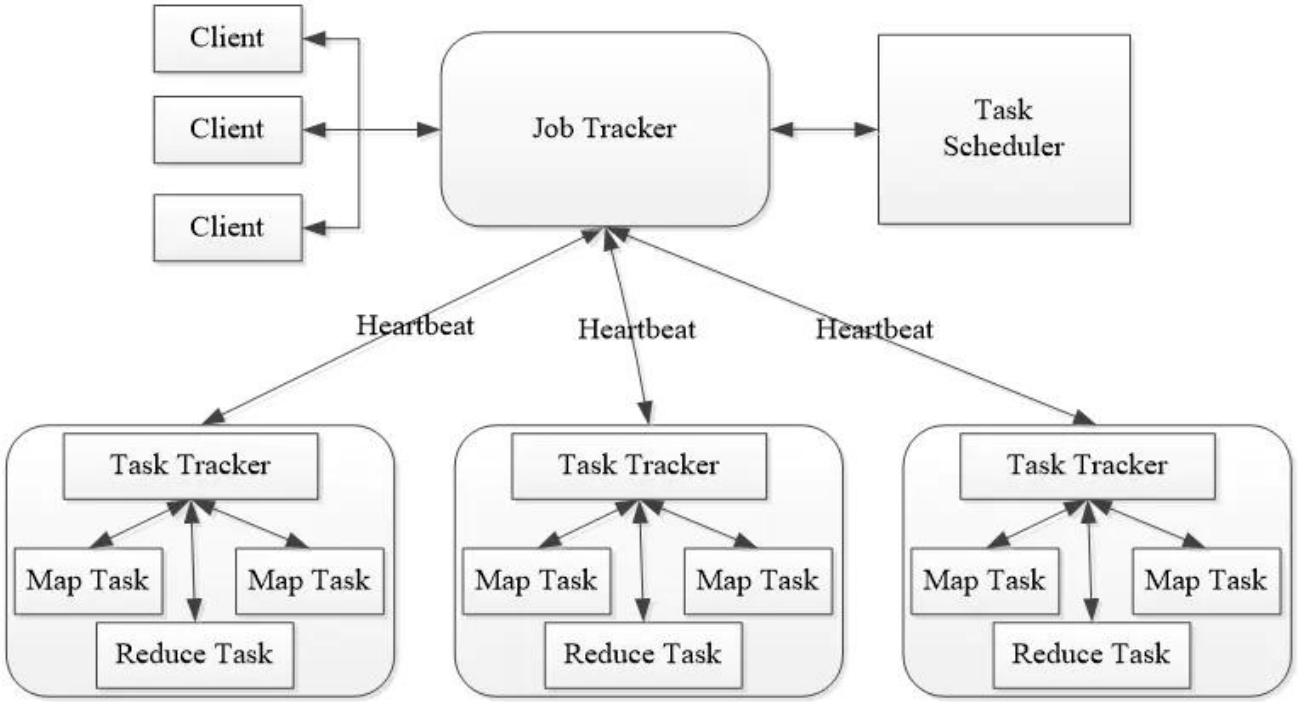
北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

MapReduce集群调度

Hadoop1.X的资源分配与任务调度

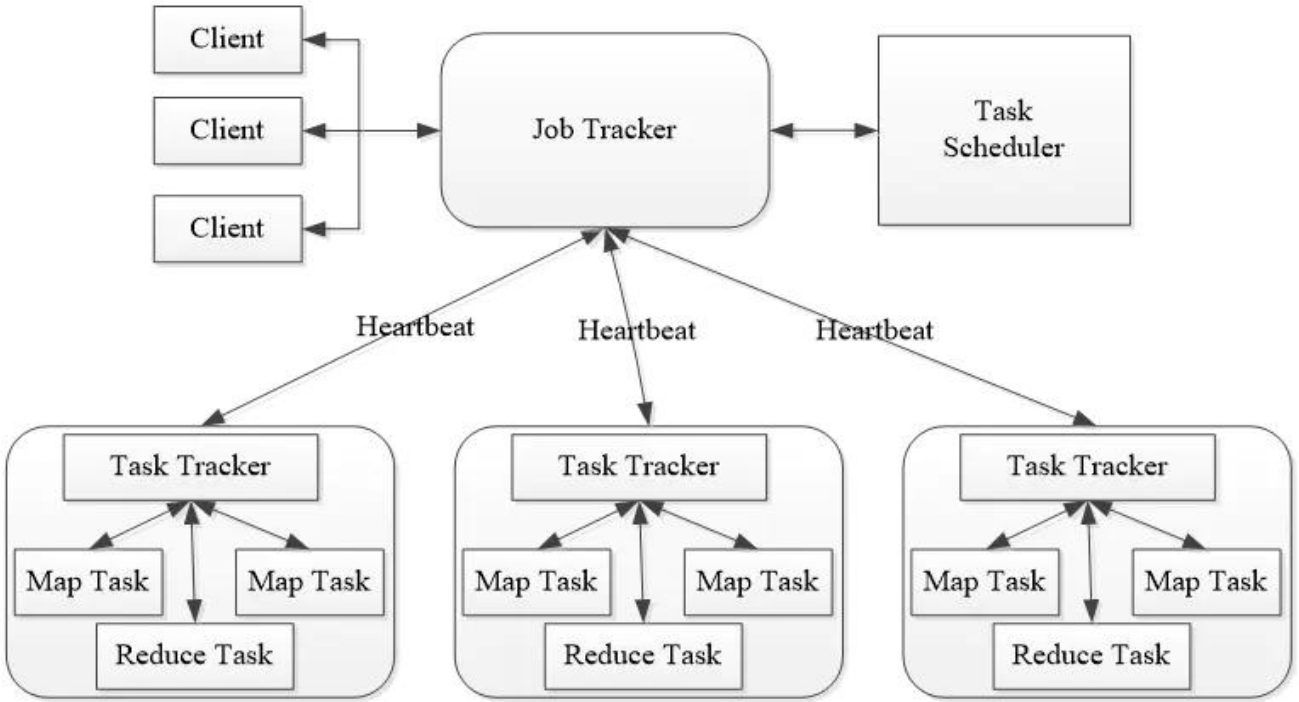
北京郵電大學 BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

- Hadoop1.x利用slots来管理分配节点的资源
- 每个Job申请资源以slots为单位，每个节点会确定自己的计算能力以及memory确定自己包含的slots总量
- 当某个Job要开始执行时，先向JobTracker申请slots，JobTracker分配空闲的slots，Job再占用slots，Job结束后，归还slots



Hadoop1.X的资源分配与任务调度

- 每个TaskTracker定期（心跳周期）通过心跳(hearbeat)与Jobtracker通信，一方面汇报自己当前工作状态，JobTracker得知某个TaskTracker是否Alive；同时汇报自身空闲slots数量
- JobTracker利用某个调度规则，如Hadoop默认调度器FIFO或者Capacity Scheduler、FairScheduler等进行任务调度



1.X版本的独立集群集中调度

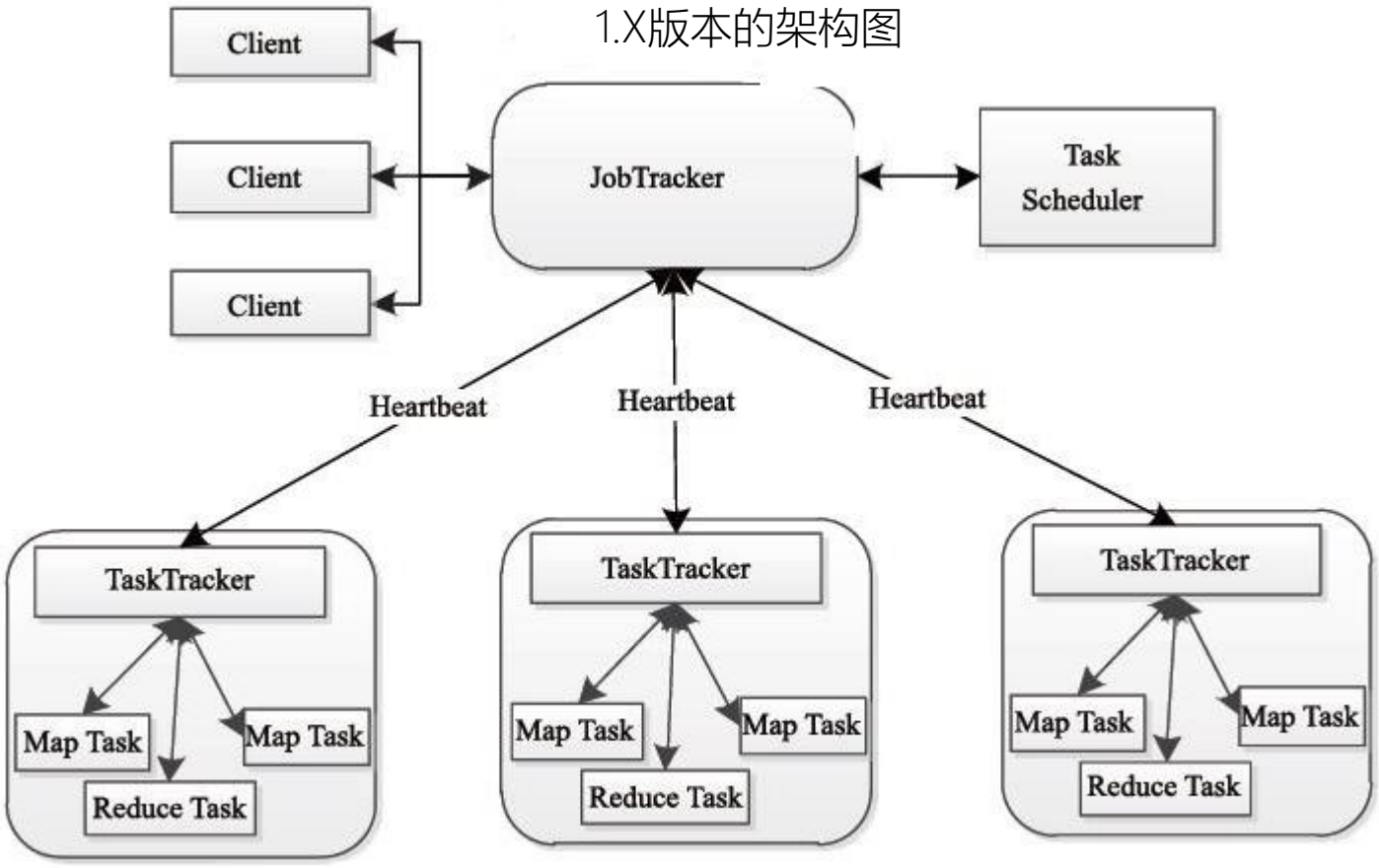
中央式调度器特点是：

- 资源调度和作业管理功能全部都放到一个进程中完成

缺陷：

- 集群的扩展性差，集群的规模容易受限
- 新的调度策略难以融入现有的代码，容易发生性能瓶颈和单点故障

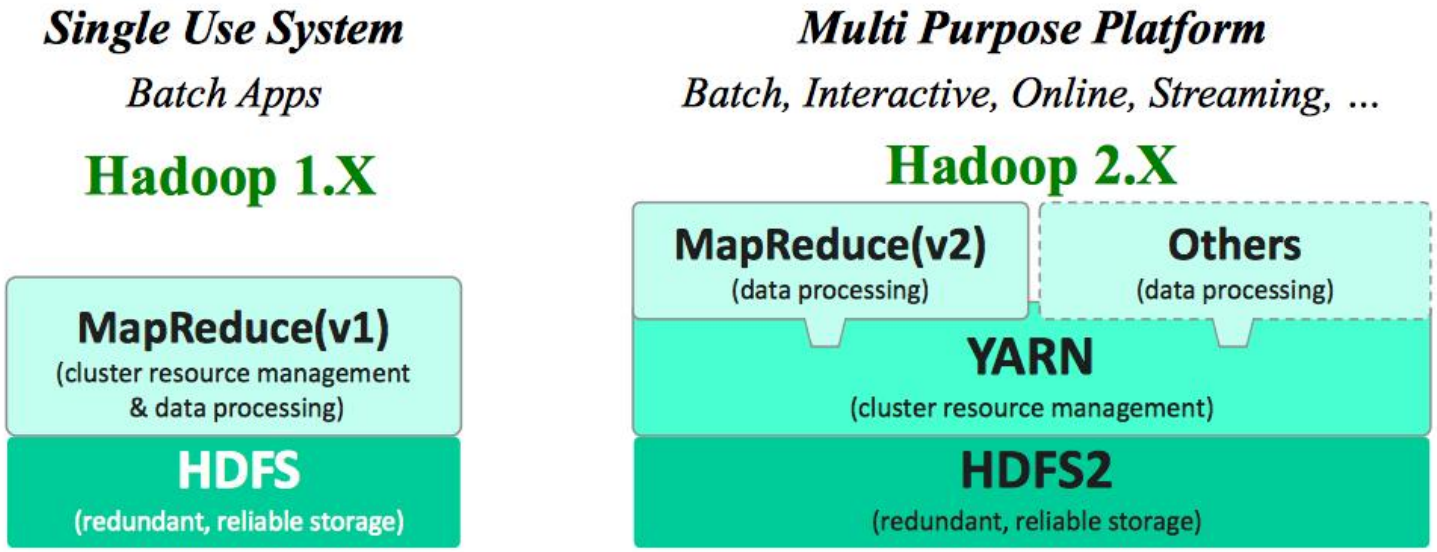
1.X版本的架构图



- ✓ JobTracker：负责资源管理和作业调度
- ✓ TaskTracker：定期向JobTracker汇报本节点的健康状况、资源使用情况、作业执行情况；接受来自JobTracker的命令:启动任务/杀死任务

Hadoop/MapReduce1.x的架构问题

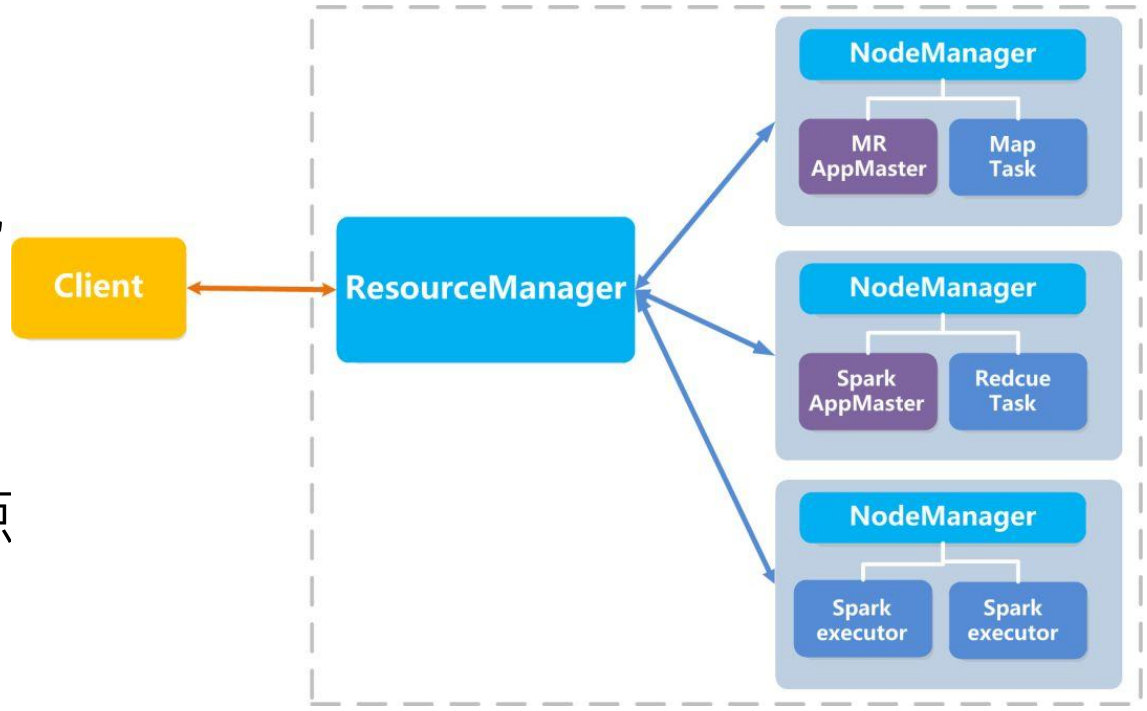
- 单点故障&节点压力大&不易扩展
  - JobTraker受内存限制，导致扩展性受限
  - 中心化架构的通病，一旦JobTraker崩溃，会导致整个集群崩溃
  - TaskTraker的Map Slot和Reduce Slot是固定的，是一种低效的静态资源分配，不是动态分配的资源



2.X版本引入双层调度架构-YARN

- YARN：由ResourceManager、NodeManager和ApplicationMaster三部分组成
- ResourceManager（RM）：主节点服务，负责维护节点信息和负责资源管理与作业调度，可以部署两台并利用Zookeeper 实现高可用
- NodeManager （NM）：计算节点服务，负责接收RM请求分配Container并管理Container的生命周期；向RM汇报所在节点信息；

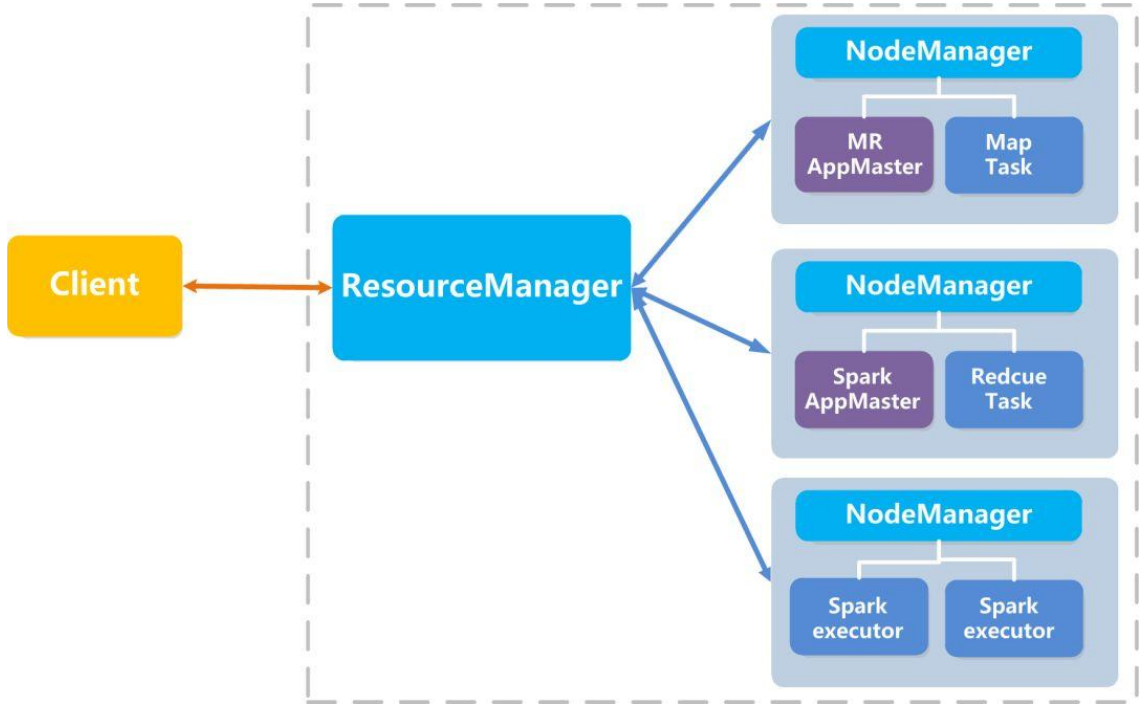
Hadoop 2.0 YARN 架构



2.X版本引入双层调度架构-YARN

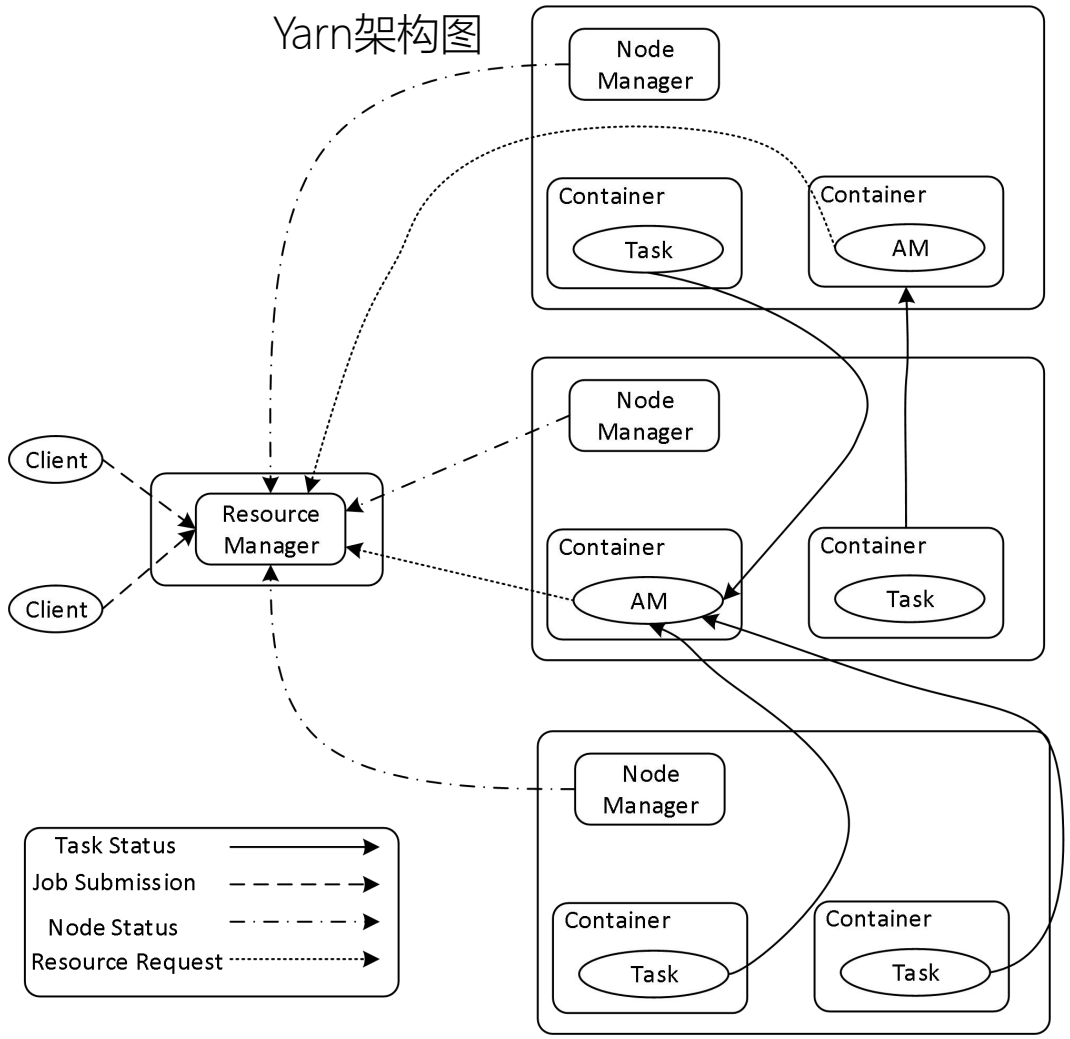
- ApplicationMaster（AM）：用户每提交一个应用都会包含一个ApplicationMaster，负责与RM通信，申请或释放资源；与NM通信启动和停止Task；监控任务的运行状态
- Client：负责提交作业，同时提供一些命令行工具

Hadoop 2.0 YARN 架构



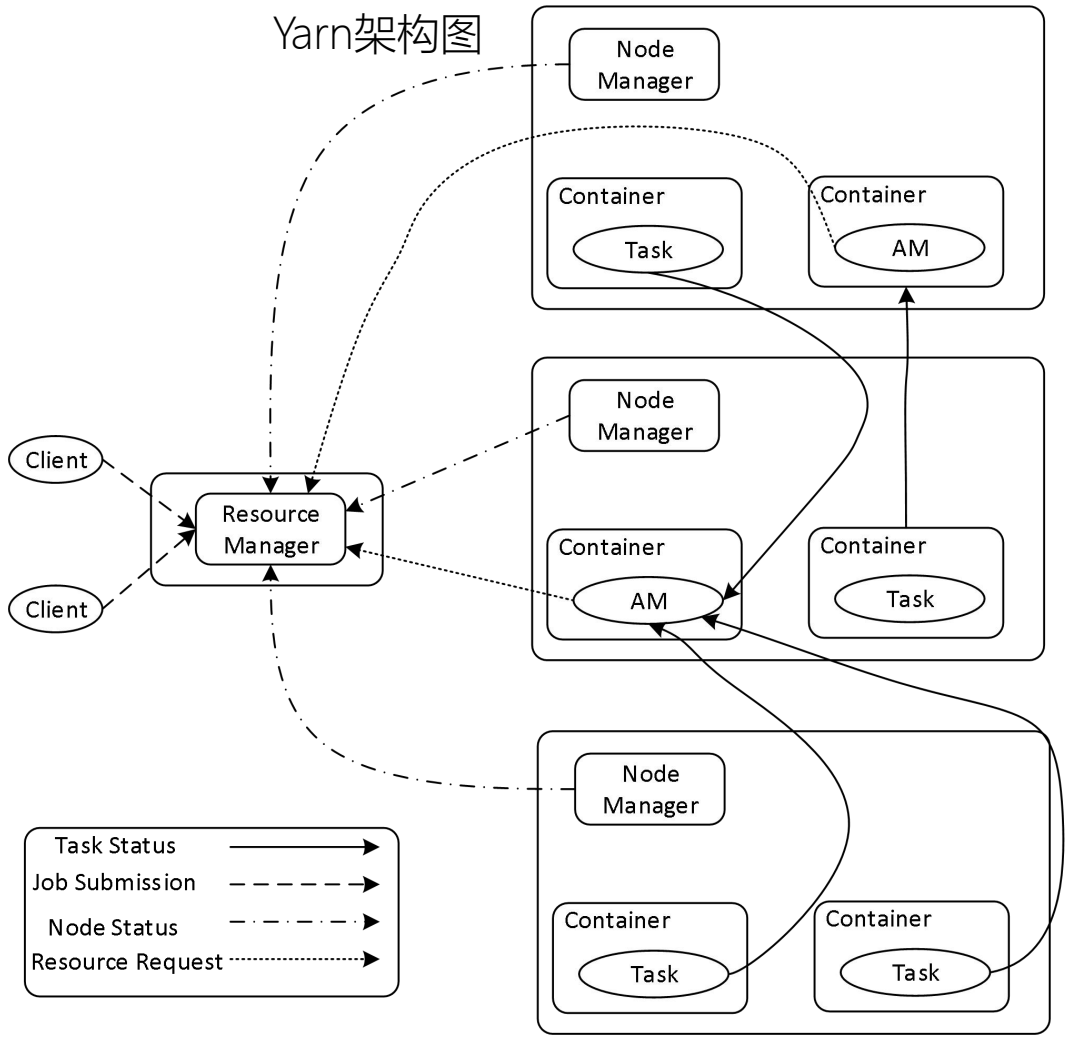
YARN的Container

- ✓ **Container**：Container是YARN中的资源抽象，它封装了多个维度的资源，如CPU、内存、磁盘、网络等
- ✓ 当 AM 向 RM 申请资源时，RM 为 AM 返回的资源是用 Container 表示的
- ✓ YARN 会为每个任务分配一个 Container，该任务只能使用该 Container 中描述的资源



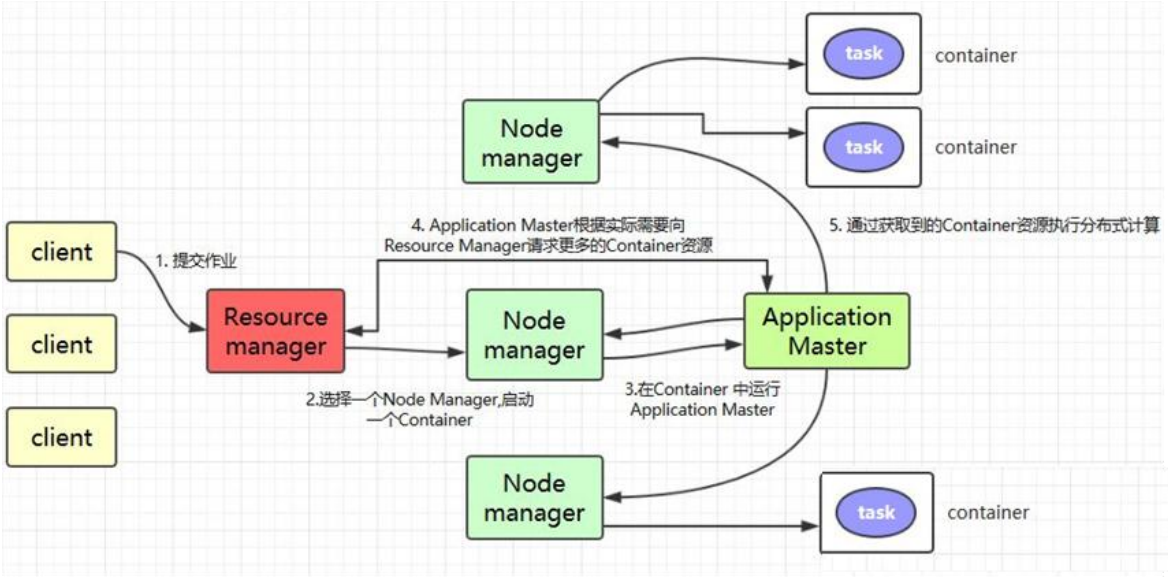
YARN的Container

- ✓ ApplicationMaster 可在 Container 内运行任何类型的任务
  - ✓ MapReduce ApplicationMaster 请求一个容器来启动 map 或 reduce 任务
  - ✓ Spark ApplicationMaster请求一个容器来运行 Spark任务



YARN的调度流程（简化）

- Client 提交作业到 YARN 上
- Resource Manager 选择一个 Node Manager，启动一个 Container 并运行 Application Master 实例
- Application Master 根据实际需要向 Resource Manager 请求更多的 Container 资源（如果作业很小, 应用管理器会选择在其自己的 JVM 中运行任务）
- Application Master 通过获取到的 Container 资源执行分布式计算



Yarn分层调度的要点

- 核心1-ResourceManager这个实体控制整个集群并管理应用程序向基础计算资源的分配
- ResourceManager
  - 管理资源调度：将各个资源部分（计算、内存、带宽等）精心安排给基础 NodeManager（YARN 的每节点代理）
  - 管理资源分配：与 ApplicationMaster 一起分配资源，与 NodeManager 一起启动和监视它们的基础应用程序。在此上下文中，ApplicationMaster 承担了以前的 TaskTracker 的一些角色，ResourceManager 承担了 JobTracker 的角色
  - 管理ApplicationMaster 生命周期

Yarn分层调度的要点

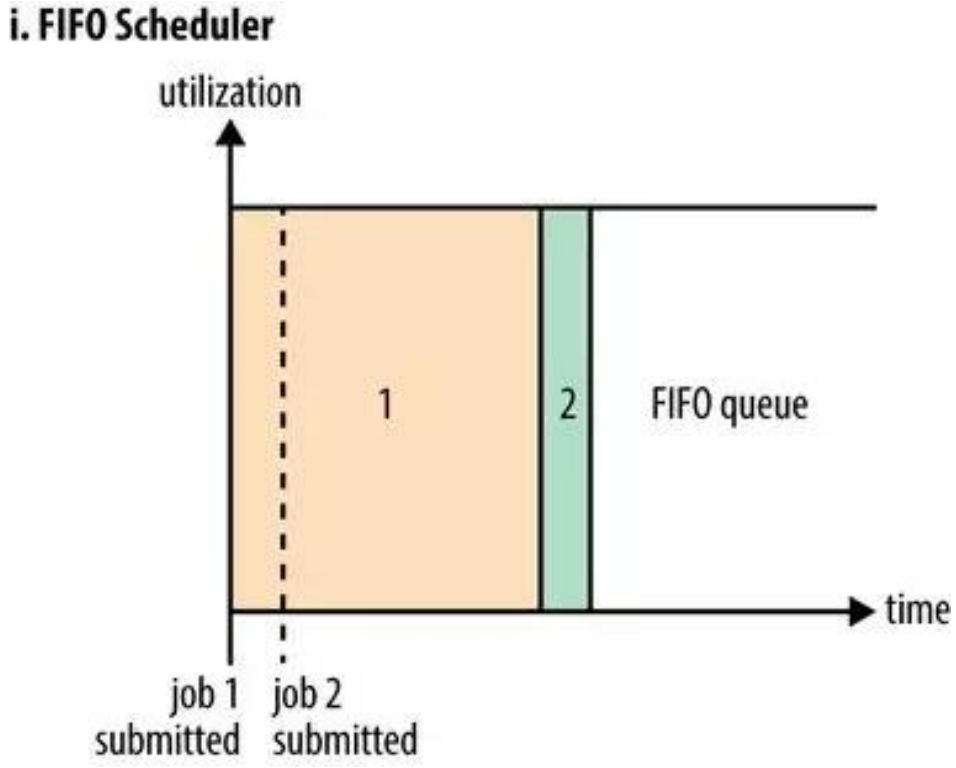
- 核心2-ApplicationMaster管理一个在 YARN 内运行的应用程序的每个实例
  - ApplicationMaster 负责协调来自 ResourceManager 的资源，并通过 NodeManager 监视容器的执行和资源使用（CPU、内存等的资源分配）。
- NodeManager 管理一个 YARN 集群中的每个节点；NodeManager 提供针对集群中每个节点的服务，监督一个容器全生命周期；监视资源和跟踪节点健康

Yarn的调度器

- 理想情况下，应用对Yarn资源的请求应该立刻得到满足
- 现实中资源往往是有限的，特别是在一个很繁忙的集群，一个应用资源的请求经常需要等待一段时间才能的到相应的资源
- 很难找到一个完美的策略可以解决所有的应用场景
- Yarn提供了多种调度器和可配置的策略供用户选择
- Yarn中负责给应用分配资源的就是Scheduler
- Yarn中提供了三种调度器：
  - FIFO Scheduler：先进先出调度器
  - Capacity Scheduler：能力/容量调度器
  - Fair Scheduler：公平调度器
- 用户也可以继承 ResourceScheduler 的接口实现自定义的调度器

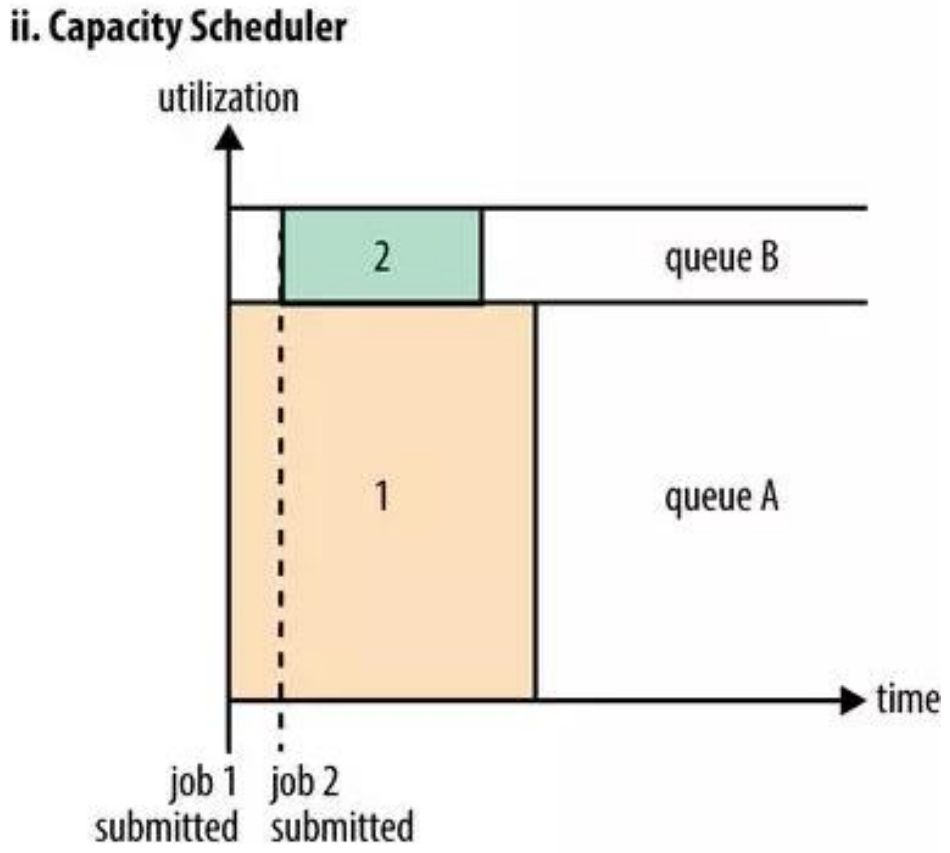
FIFO Scheduler

- 一个的简单调度器，适合低负载集群
- 把应用按提交的顺序排成一个队列，这是一个先进先出队列
- 在进行资源分配的时候，先给队列中最头上的应用进行分配资源，待最头上的应用需求满足后再给下一个分配，以此类推



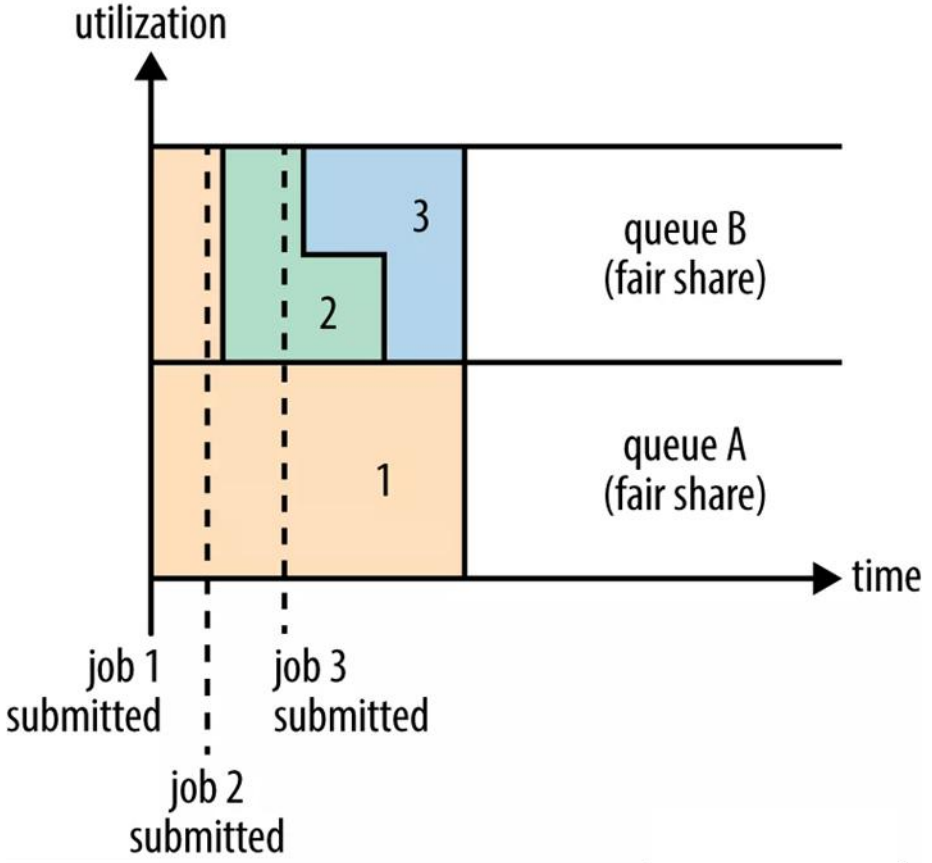
Capacity Scheduler

- **用户共享集群能力**：支持多用户共享集群和多应用程序同时运行，每个组织可以获得集群的一部分计算能力；防止单个应用程序、用户或者队列独占集群中的资源
- 给不同队列（即用户或用户组）分配一个预期最小容量，在每个队列内部用层次化的FIFO来调度多个应用程序。
- 为了避免某个队列过多的占用空闲资源，导致其它队列无法使用这些空闲资源可以为队列设置一个最大资源使用量
- **弹性队列(queue elasticity)**：在正常的操作中，Capacity调度器不会强制释放Container，当一个队列资源不够用时，这个队列能获得其它队列释放后的Container资源



Fair Scheduler：公平调度器

- Fair调度器比较适用于多用户共享的大集群，设计目标是为所有的应用分配公平的资源（对公平的定义可以通过参数来设置）
- 当只有一个 job 在运行时，该应用程序最多可获取所有资源，再提交其他 job 时，资源将会被重新分配分配给目前的 job，这可以让大量 job 在合理的时间内完成，减少作业 pending 的情况
  - 假设有两个用户A和B，他们分别拥有一个队列
  - 当A启动一个job而B没有任务时，A会获得全部集群资源
  - 当B启动一个job后，A的job会继续运行，不过一会儿之后两个任务会各自获得一半的集群资源
  - 如果此时B再启动第二个job并且其它job还在运行，则它将会和B的第一个job共享B这个队列的资源，也就是B的两个job会用于四分之一的集群资源，而A的job仍然用于集群一半的资源
  - 结果就是资源最终在两个用户之间平等的共享



抢占（Preemption）操作

- 当一个job提交到一个繁忙集群中的空队列时，job并不会马上执行，而是阻塞直到正在运行的job释放系统资源
- 为了使提交job的执行时间更具预测性（可以设置等待的超时时间），Fair调度器支持抢占
- 抢占就是允许调度器杀掉占用超过其应占份额资源队列的containers，这些containers资源便可被分配到应该享有这些份额资源的队列中
- 需要注意抢占会降低集群的执行效率，因为被终止的containers需要被重新执行
- 可以通过设置一个全局的参数yarn.scheduler.fair.preemption=true来启用抢占功能

谢谢聆听
批评指正

ehaihong@bupt.edu.cn