厦门大学计算机科学系研究生课程

《大数据技术基础》

第4章 MapReduce
（2013年新版）

林子雨

厦门大学计算机科学系

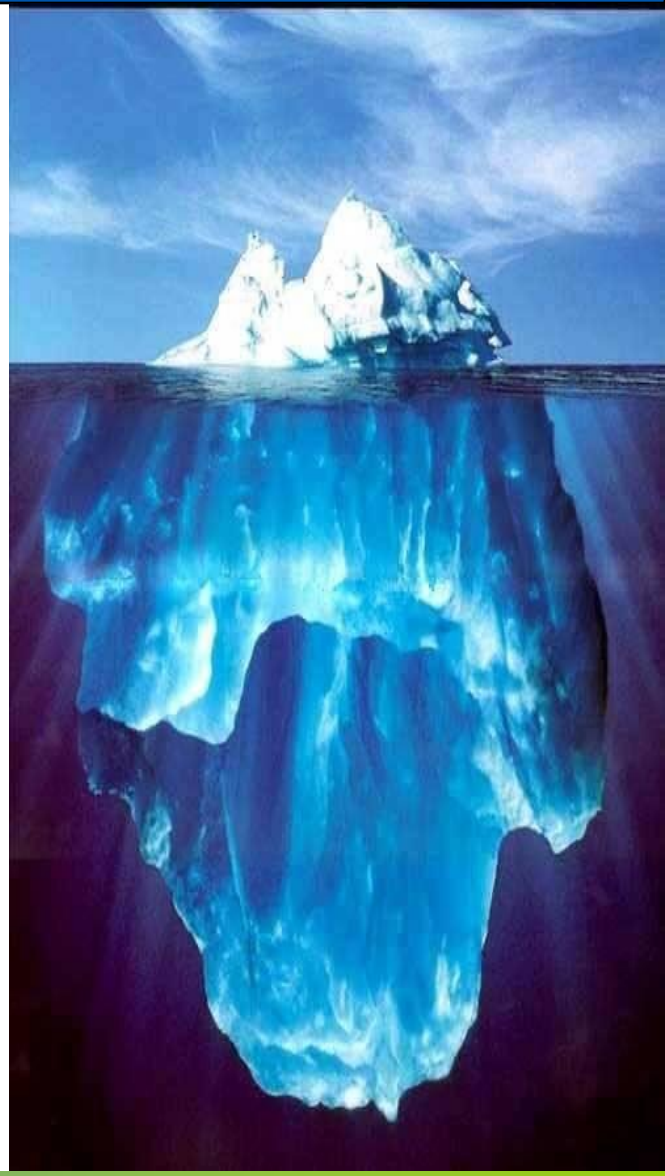
E-mail: ziyulin@xmu.edu.cn

个人主页：http://www.cs.xmu.edu.cn/linziyu

课程提要

- 分布式并行编程：编程方式的变革
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn

本讲义PPT存在配套教材，由林子雨通过大量阅读、收集、整理各种资料后编写而成
下载配套教材请访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423



分布式并行编程：编程方式的变革

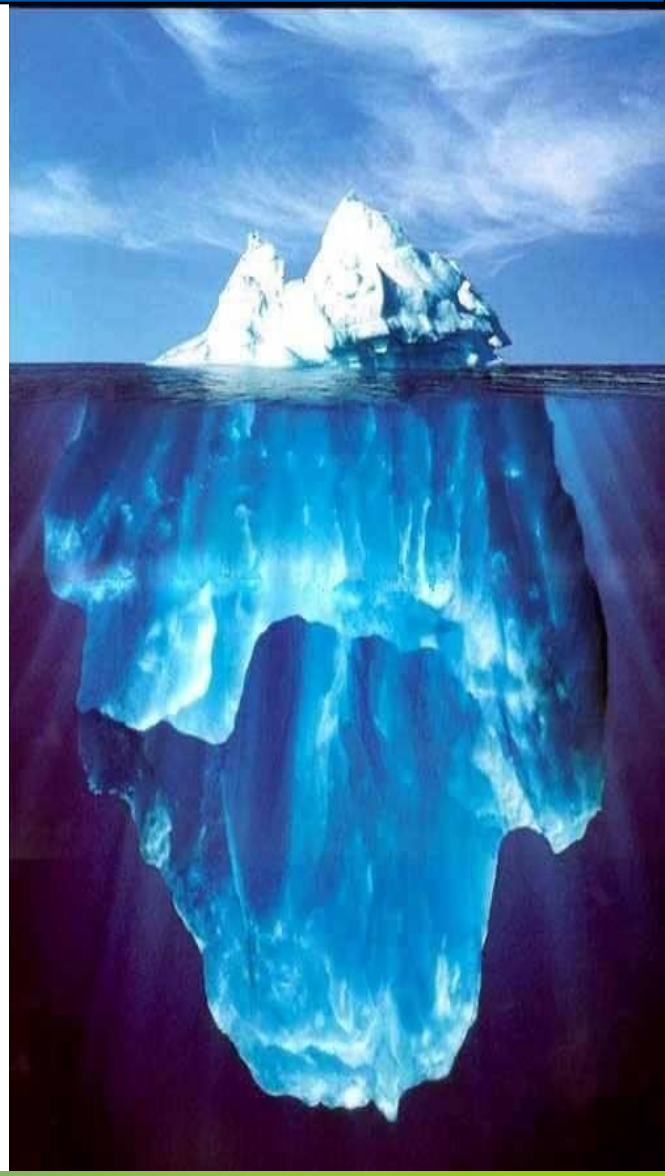
- 根据摩尔定律，约每隔18个月，CPU性能会提高一倍。在摩尔定律的作用下，软件不用做任何改变，就可以享受性能的提升。
- 然而，由于晶体管电路已经逐渐接近其物理上的性能极限，摩尔定律在 2005 年左右开始失效了，人类再也不能期待单个 CPU 的速度每隔 18 个月就翻一倍，为我们提供越来越快的计算性能。
- Intel、AMD、IBM等芯片厂商开始从多核这个角度来挖掘CPU的性能潜力，多核时代以及互联网时代的到来，将使软件编程方式发生重大变革，基于多核的多线程并发编程以及基于大规模计算机集群的分布式并行编程是将来软件性能提升的主要途径。

分布式并行编程：编程方式的变革

- 基于集群的分布式并行编程，能够让软件与数据同时运行在连成一个网络的许多台计算机上，可以很容易地通过增加计算机来扩充新的计算结点，并由此获得不可思议的海量计算能力，但分布式并行开发与传统的顺序执行开发逻辑大不相同。
- 开源的 Hadoop的出现，则极大地降低了分布式并行开发的门槛。基于 Hadoop编程非常简单，无需任何并行开发经验，也可以轻松地开发出分布式的并行程序，并让其令人难以置信地同时运行在数百台机器上，然后在短时间内完成海量数据的计算。
- 随着“云计算”的普及，任何人都可以轻松获得这样的海量计算能力。例如，现在 Amazon 公司的云计算平台 Amazon EC2 已经提供了这种按需计算的租用服务。

课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



MapReduce模型概述

- MapReduce是Google公司的核心计算模型，它将复杂的运行于大规模集群上的并行计算过程高度地抽象到两个函数：Map和Reduce。
- 适合用MapReduce来处理的数据集(或任务)，需要满足一个基本要求：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。
- 概念“Map（映射）”和“Reduce（化简）”，以及它们的主要思想，都是从函数式编程语言里借来的，同时包含了从矢量编程语言里借来的特性。MapReduce极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。

MapReduce模型概述

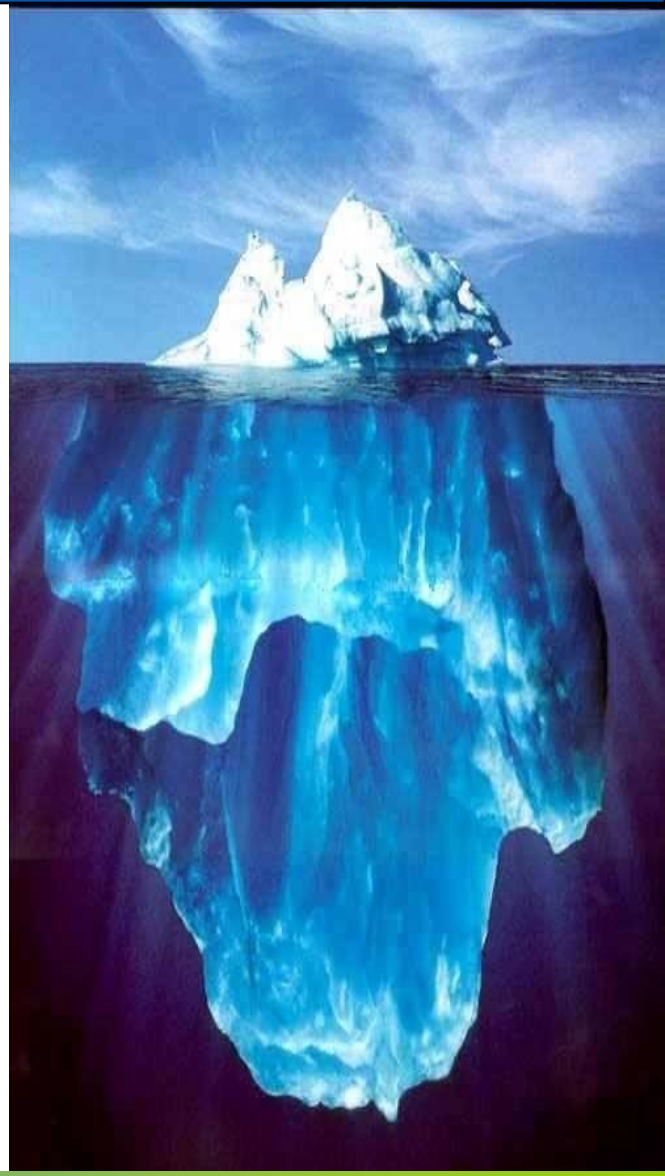
- 一个Map-Reduce作业（job）通常会把输入的数据集切分为若干独立的数据块，由map任务（task）以完全并行的方式处理它们。框架会对map的输出先进行排序，然后把结果输入给reduce任务。通常作业的输入和输出都会被存储在文件系统中。整个框架负责任务的调度和监控，以及重新执行已经失败的任务。
- Map-Reduce框架和分布式文件系统是运行在一组相同的节点上的，即计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。
- Map-Reduce框架由单独一个master JobTracker和每个集群节点一个slave TaskTracker共同组成。这个master负责调度构成一个作业的所有任务，这些任务分布在不同的slave上，master监控它们的执行，重新执行已经失败的任务。而slave仅负责执行由master指派的任务。

MapReduce模型概述

- 应用程序至少应该指明输入/输出的位置（路径），并通过实现合适的接口或抽象类提供map和reduce函数。再加上其他作业的参数，就构成了作业配置（job configuration）。然后，由Hadoop的job-client提交作业（jar包/可执行程序等）和配置信息给JobTracker，后者负责分发这些软件和配置信息给slave、调度任务且监控它们的执行，同时提供状态和诊断信息给job-client。
- 虽然Hadoop框架是用Java实现的，但Map-Reduce应用程序则不一定要用Java来写 。

课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



Map和Reduce函数

- MapReduce计算模型的核心是map和reduce两个函数，这两个函数由用户负责实现，功能是按一定的映射规则将输入的<key，value>对转换成另一个或一批<key，value>对输出。

函数	输入	输出	说明
Map	<k1, v1>	List(<k2, v2>)	1. 将小数据集进一步解析成一批<key, value>对，输入Map函数中进行处理。 2. 每一个输入的<k1, v1>会输出一批<k2, v2>。<k2, v2>是计算的中间结果
Reduce	<k2, List(v2)>	<k3, v3>	输入的中间结果<k2, List(v2)>中的List(v2)表示是一批属于同一个k2的value

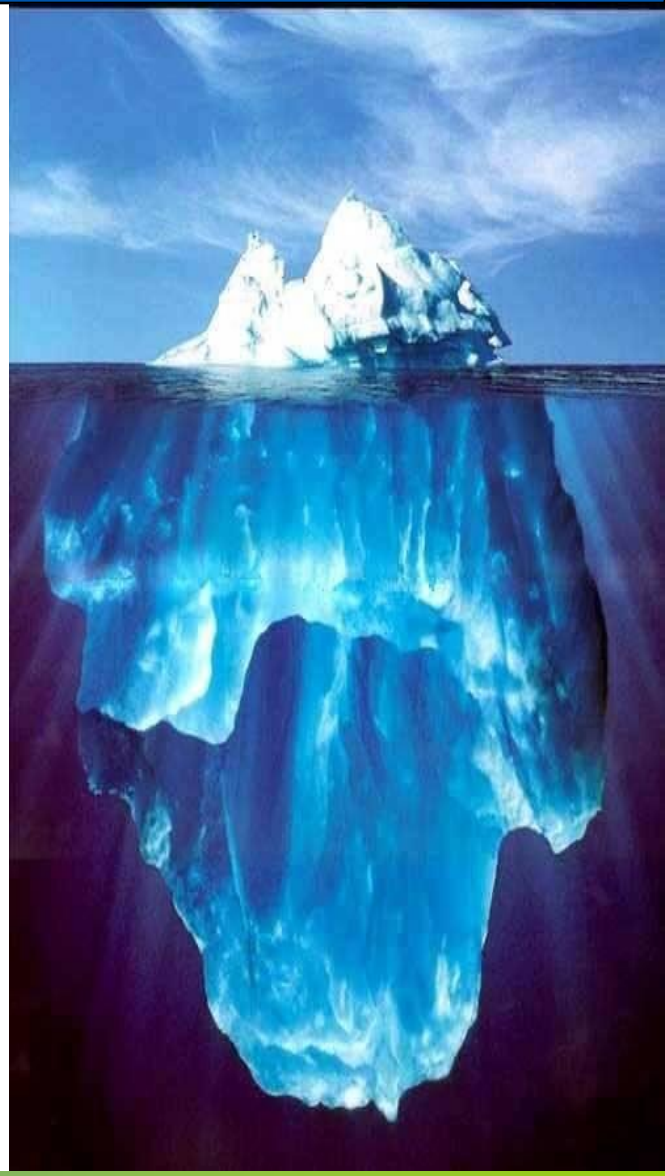
- 以计算文本文件中每个单词出现次数的程序为例，则<k1，v1>可以是<行在文件中的偏移位置，文件中的一行>，经 Map 函数映射之后，形成一批中间结果 <单词，出现次数>， 而 Reduce 函数则可以对中间结果进行处理，将相同单词的出现次数进行累加，得到每个单词的总的出现次数。

Map和Reduce函数

- 基于MapReduce计算模型编写分布式并行程序非常简单，程序员的主要编码工作就是实现Map和Reduce函数，其它的并行编程中的种种复杂问题，如分布式存储、工作调度、负载平衡、容错处理、网络通信等，均由MapReduce框架(比如 Hadoop)负责处理，程序员完全不用操心。

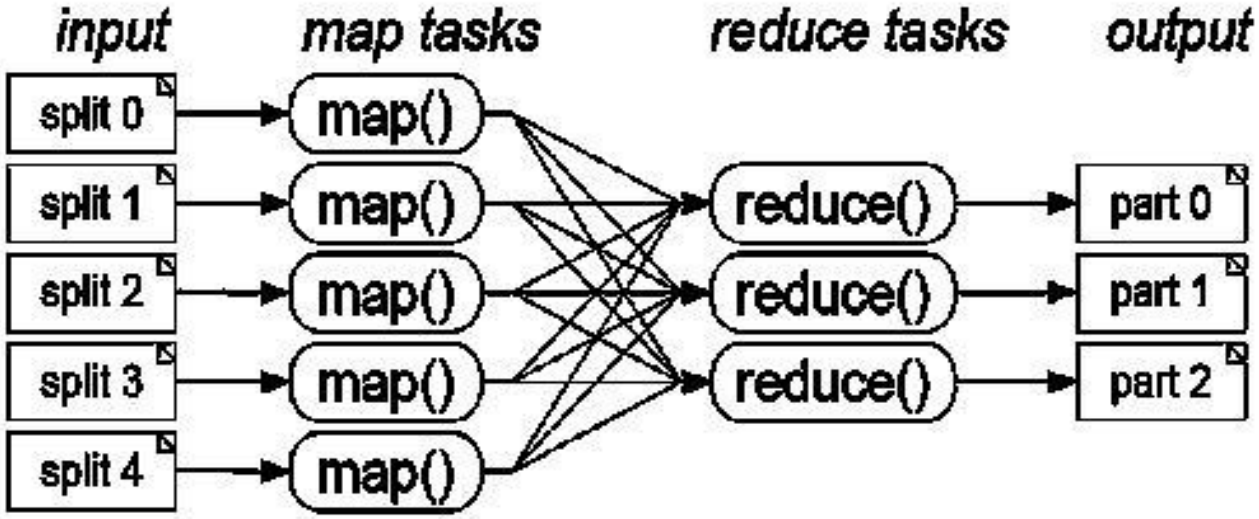
课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



MapReduce工作流程概述

- 下图说明了用MapReduce来处理大数据集的过程，就是将大数据集分解为成百上千的小数据集，每个(或若干个)数据集分别由集群中的一个结点(一般就是一台普通的计算机)进行处理并生成中间结果，然后这些中间结果又由大量的结点进行合并，形成最终结果。



MapReduce工作流程概述

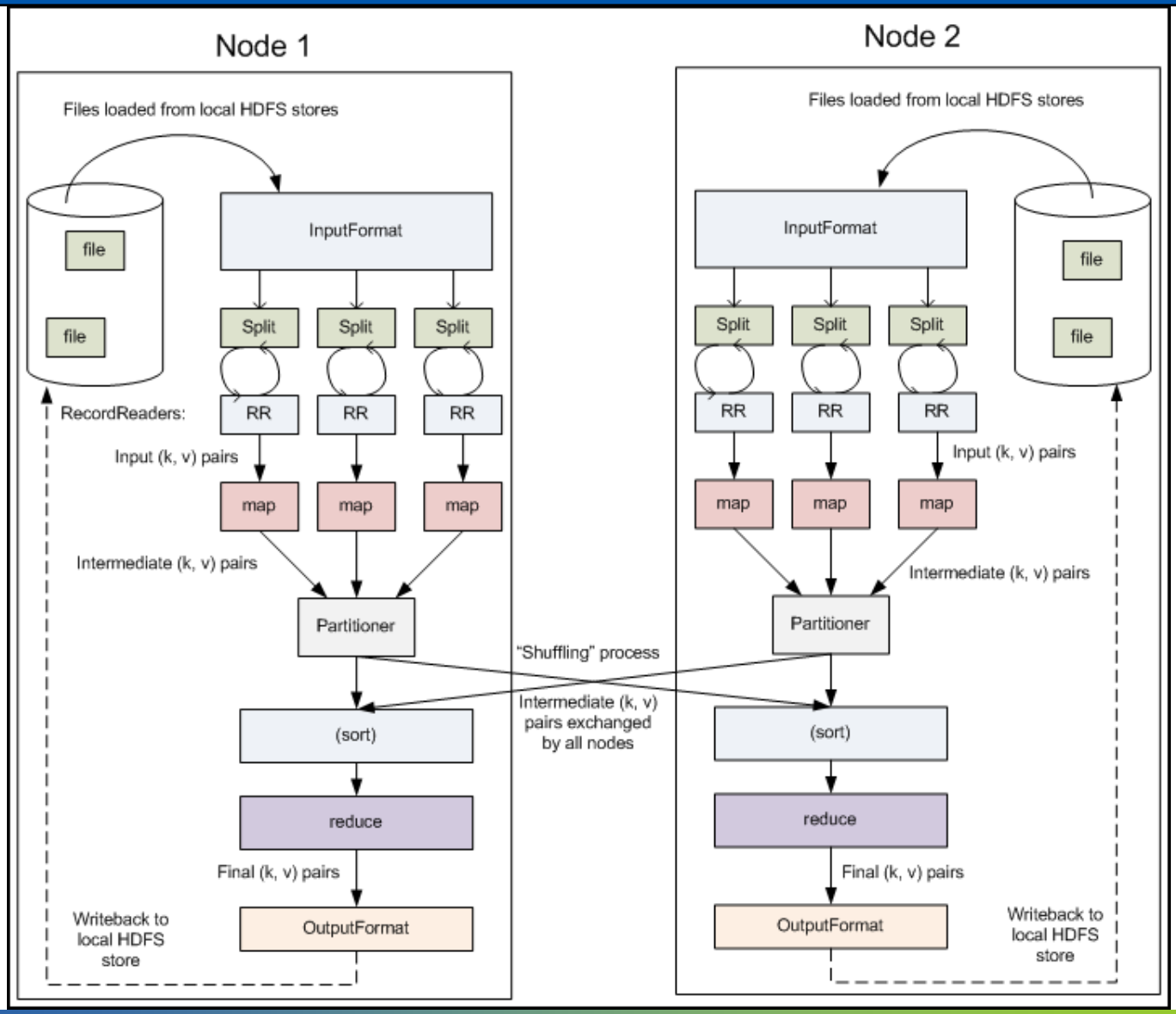
- MapReduce的输入一般来自HDFS中的文件，这些文件分布存储在集群内的节点上。运行一个MapReduce程序会在集群的许多节点甚至所有节点上运行mapping任务，任意的mapper都可以处理任意的输入文件。
- 当mapping阶段完成后，这阶段所生成的中间键值对数据必须在节点间进行交换，把具有相同键的数值发送到同一个reducer。Reduce任务在集群内的分布节点同mappers的一样。这是MapReduce中唯一的任务节点间的通信过程。
- 所有数据传送都是由Hadoop MapReduce平台自身去做的，是通过关联到数值上的不同键来隐式引导的。这是Hadoop MapReduce的可靠性的基础元素。如果集群中的节点失效了，任务必须可以被重新启动。

MapReduce各个执行阶段

- 一般而言，Hadoop的一个简单的MapReduce任务执行流程如下：

1) JobTracker负责分布式环境中实现客户端创建任务并提交。
2) InputFormat模块负责做Map前的预处理。
3) 将RecordReader处理后的结果作为Map的输入，然后Map执行定义的Map逻辑，输出处理后的(key,value)对到临时中间文件。
4) Shuffle&Partitioner：在MapReduce流程中，为了让reduce可以并行处理map结果，必须对map的输出进行一定的排序和分割，然后再交给对应的reduce。这个将map输出进行进一步整理并交给reduce的过程，就称为shuffle。Partitioner是选择配置，主要作用是在多个Reduce的情况下，指定Map的结果由某一个Reduce处理，每一个Reduce都会有单独的输出文件。
5) Reduce执行具体的业务逻辑，即处理数据以得到结果的业务，并且将处理结果输出给OutputFormat。
6) OutputFormat的作用是，验证输出目录是否已经存在和输出结果类型是否符合Config中配置类型，若成立则输出Reduce汇总后的结果。

MapReduce执行阶段

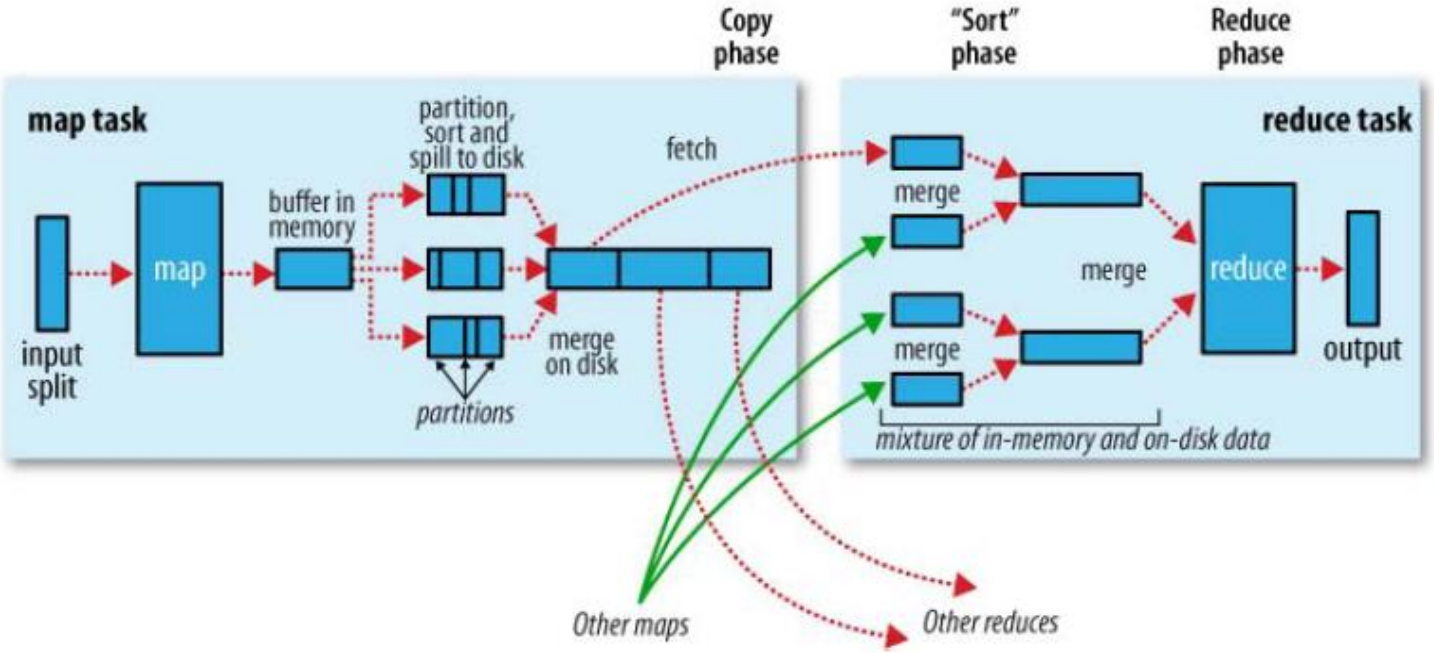


Shuffle过程详解

- Shuffle：将Map输出进行进一步整理并交给reduce的过程。
- Shuffle过程是MapReduce工作流程的核心，也被称为奇迹发生的地方。要想理解MapReduce，Shuffle是必须要了解的。
- Shuffle过程包含在map和reduce两端中，描述着数据从map task输出到reduce task输入的这段过程。

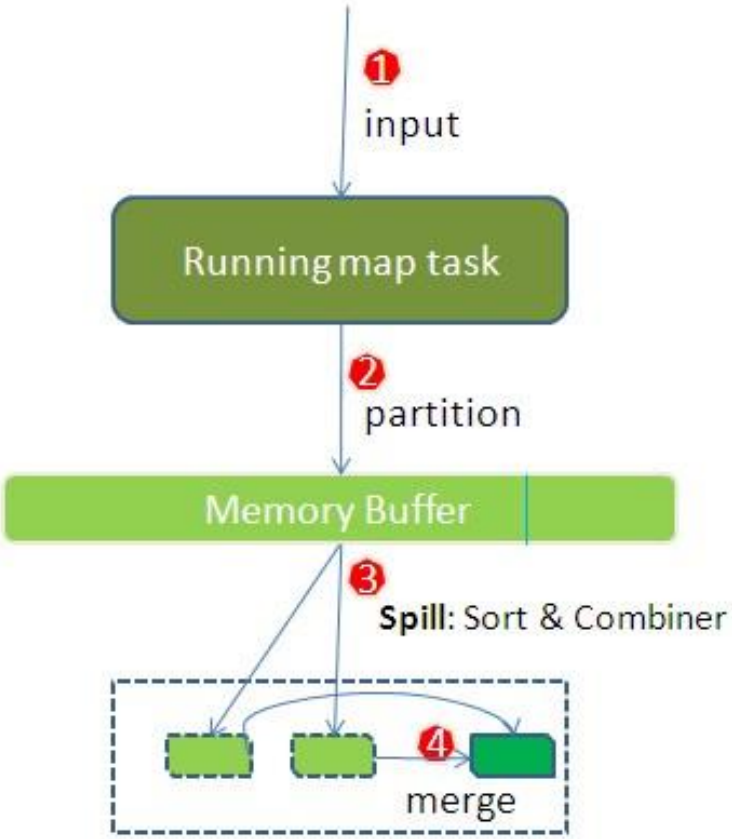
Shuffle过程详解

- 在map端的shuffle过程是对map的结果进行划分（partition）、排序（sort）和spill（溢写），然后将属于同一个划分的输出合并在一起，并写到磁盘上，同时按照不同的划分将结果发送给对应的reduce（map输出的划分与reduce的对应关系由JobTracker确定）。Reduce端又会将各个map送来的属于同一个划分的输出进行合并（merge），然后对merge的结果进行排序，最后交给reduce处理。



Shuffle过程详解 – Map端

- 下图是某个假想的map task的运行情况，可以清楚地说明划分（partition），排序（sort）与合并（combiner）作用在MapReduce工作流程的哪个阶段。



Shuffle过程详解 – Map端

- 简单地说，每个map task都有一个内存缓冲区，存储着map的输出结果，当缓冲区快满的时候，需要将缓冲区的数据以一个临时文件的方式存放到磁盘，当整个map task结束后，再对磁盘中这个map task产生的所有临时文件做合并，生成最终的正式输出文件，然后等待reduce task来拉数据。

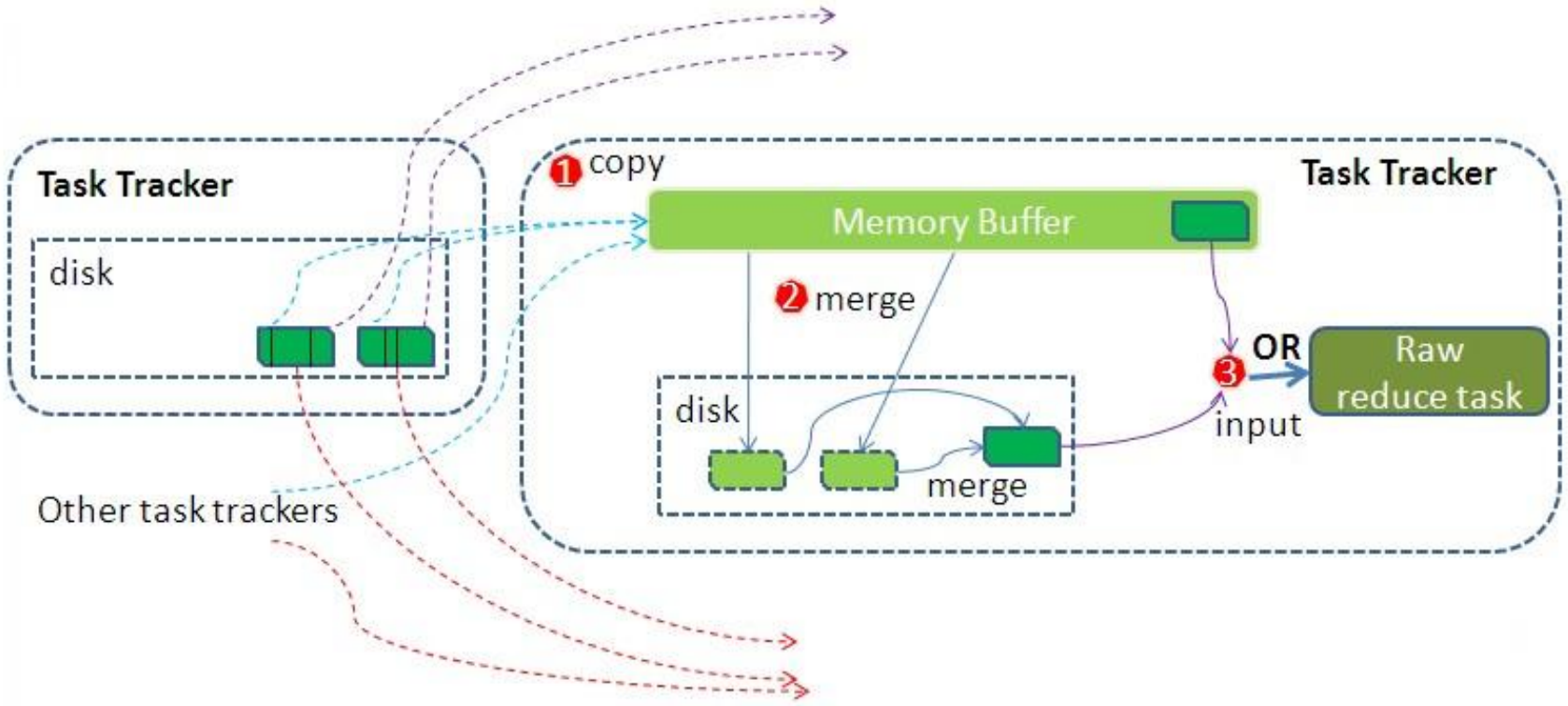
Shuffle过程详解 – Map端

- Map端的shuffle流程可分为四个步骤：

1. Map task执行：它的输入数据来源于HDFS的block
2. Mapper运行：mapper的输出是一个key/value对。
3. Spill：内存缓冲区是有大小限制的（默认是100MB）。当map task的输出结果很多时，就可能会撑爆内存，所以需要在一定条件下将缓冲区中的数据临时写入磁盘，然后重新利用这块缓冲区。这个从内存往磁盘写数据的过程被称为Spill。
4. Merge：每次溢写会在磁盘上生成一个溢写文件，如果map的输出结果真的很大，有多次这样的溢写发生，磁盘上相应的就会有多个溢写文件存在。当map task真正完成时，内存缓冲区中的数据也全部溢写到磁盘中形成一个溢写文件。最终，磁盘中会至少有一个这样的溢写文件存在。因为最终的文件只有一个，所以需要将这些溢写文件归并到一起，这个过程就叫做Merge。

Shuffle过程详解 – Reduce端

- 当map task执行完成，Shuffle的后半段过程开始启动。
- 简单地说，reduce task在执行之前的工作就是不断地拉取当前job里每个map task的最终结果，然后对从不同地方拉取过来的数据不断地做merge，也最终形成一个文件作为reduce task的输入文件。



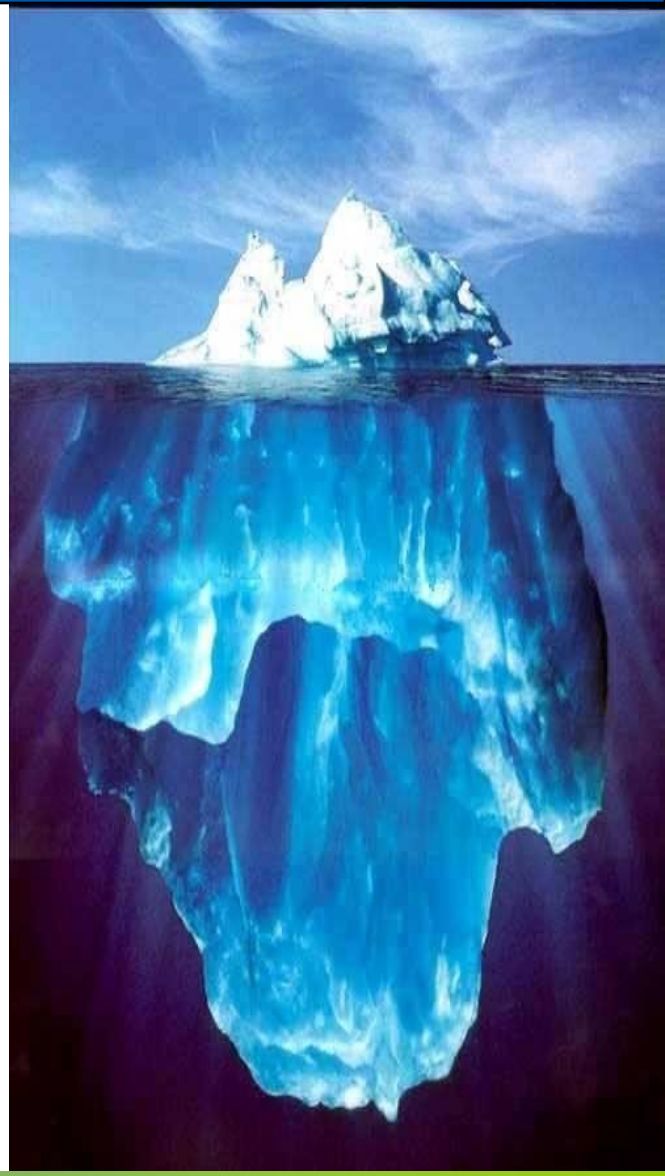
Shuffle过程详解 – Reduce端

- 当前reduce copy数据的前提是，它要从JobTracker获得有哪些map task已执行结束。Reducer真正运行之前，所有的时间都是在拉取数据，做merge，且不断重复地在做。
- 下面分成3步描述reduce端的Shuffle细节。

1. Copy过程，简单地拉取数据。Reduce进程启动一些数据copy线程(Fetcher)，通过HTTP方式请求map task所在的TaskTracker获取map task的输出文件。因为map task早已结束，这些文件就归TaskTracker管理在本地磁盘中。
2. Merge阶段。这里的merge如map端的merge动作，只是数组中存放的是不同map端copy来的数值。
3. Reducer的输入文件。不断地merge后，最后会生成一个“最终文件”（这个文件可能存在于磁盘上，也可能存在于内存中）。当Reducer的输入文件已定，整个Shuffle才最终结束。然后就是Reducer执行，把结果放到HDFS上。

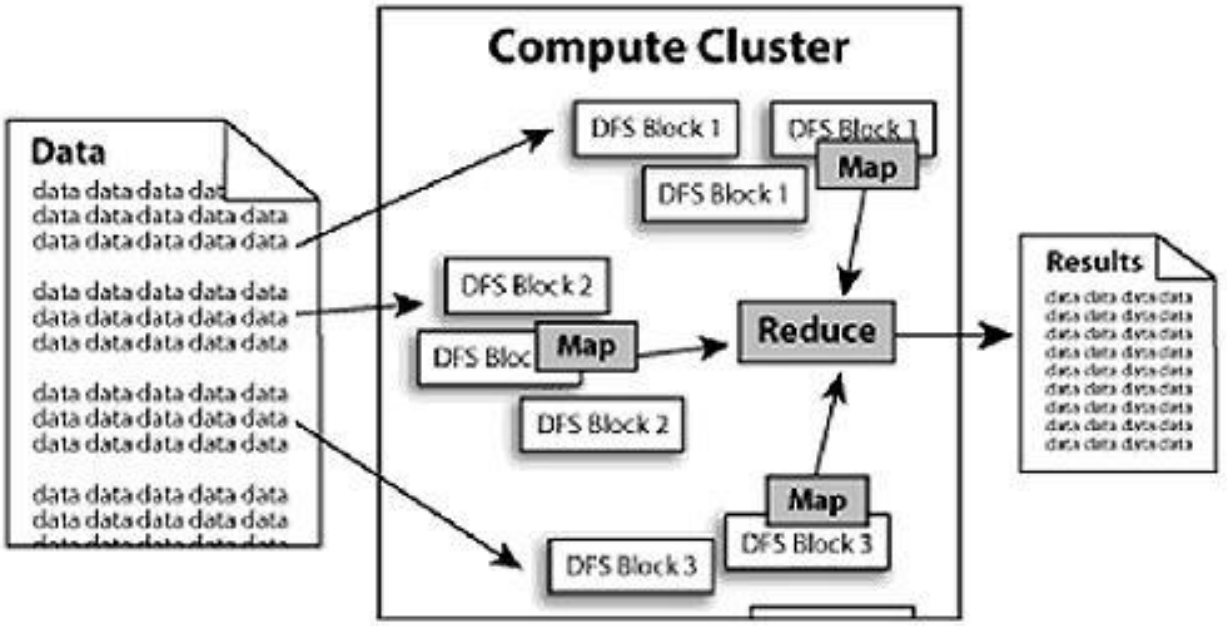
课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



并行计算的实现

- MapReduce计算模型非常适合在大量计算机组成的大规模集群上并行运行。下图每一个Map任务和每一个Reduce 任务均可以同时运行于一个单独的计算结点上，且运行效率高。这里涉及到了三个方面的内容：数据分布存储、分布式并行计算和本地计算。这三者共同合作，完成并行分布式计算的任务。

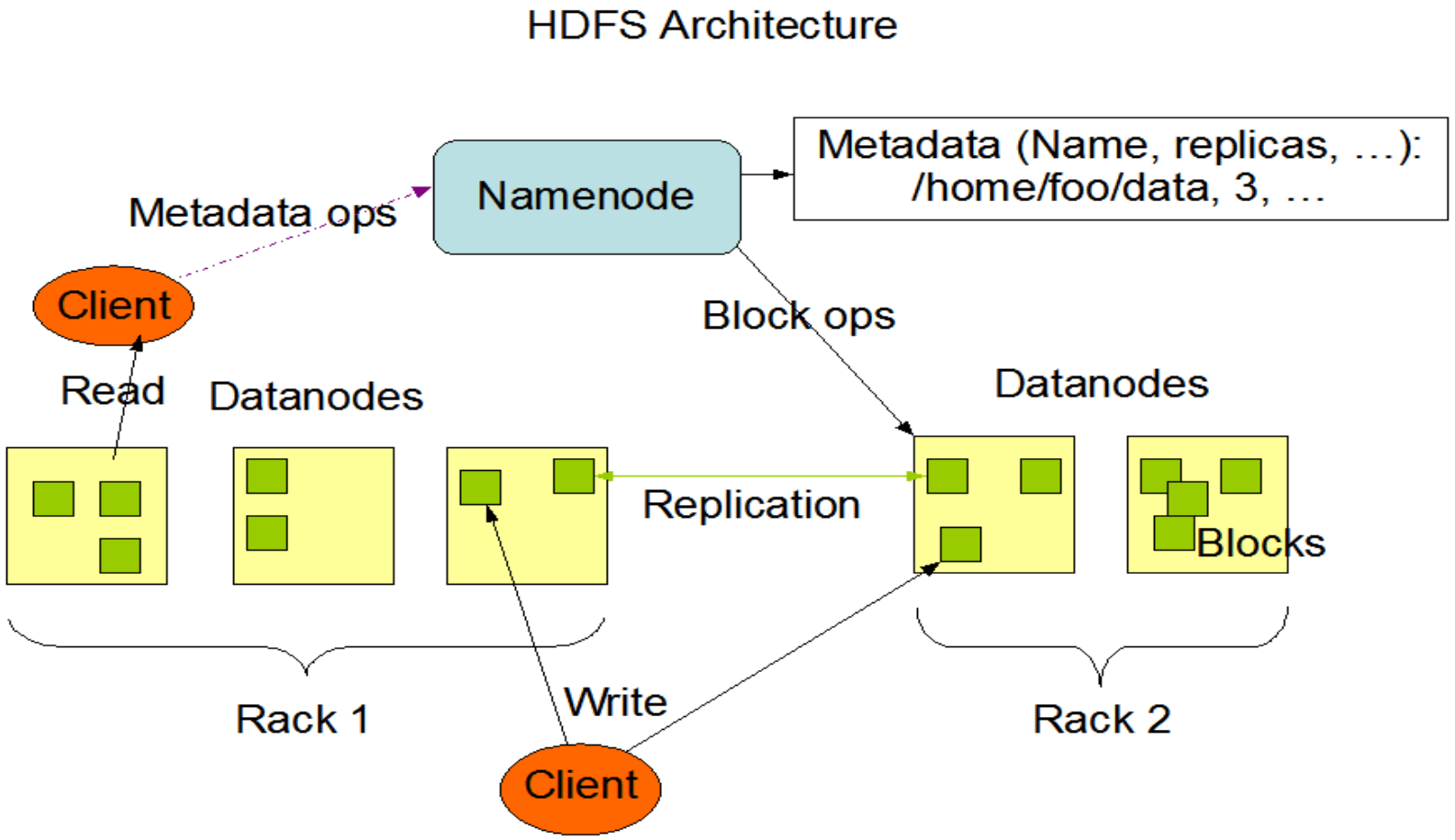


数据分布存储

- Hadoop中的分布式文件系统HDFS由一个管理结点(NameNode)和N个数据结点(DataNode)组成，每个结点均是一台普通的计算机。
- 在使用上同我们熟悉的单机上的文件系统非常类似，一样可以建目录、创建、复制、删除文件、查看文件内容等。
- HDFS其底层实现上是把文件切割成Block，然后这些Block分散地存储于不同的DataNode上，每个Block还可以复制数份存储于不同的DataNode上，达到容错容灾之目的。NameNode则是整个HDFS的核心，它通过维护一些数据结构，记录了每一个文件被切割成了多少个Block，这些Block可以从哪些DataNode中获得，各个DataNode的状态等重要信息。

数据分布存储

- 下图展示了HDFS的体系结构：

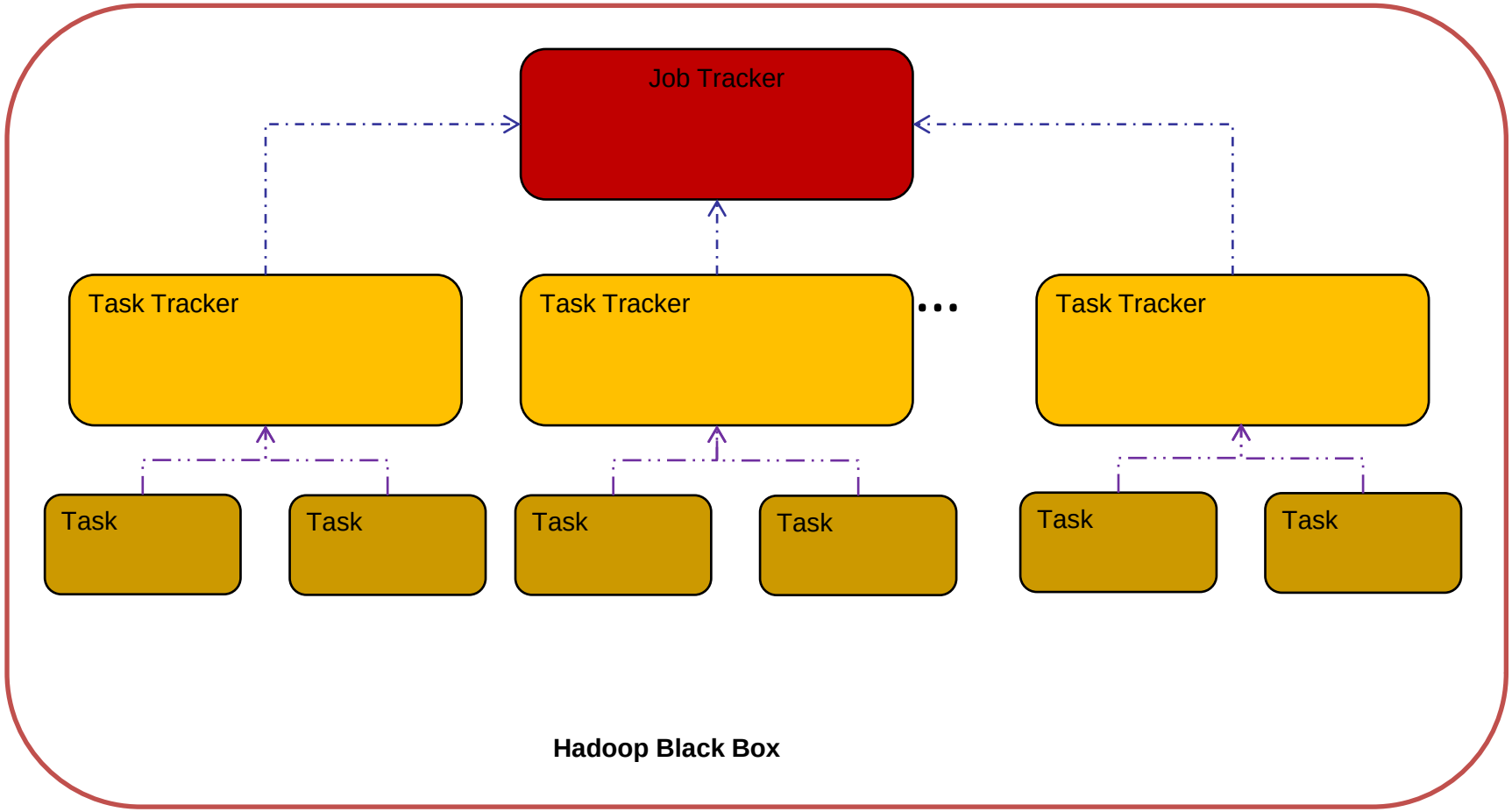


分布式并行计算

- Hadoop中有一个作为主控的JobTracker，用于调度和管理其它的TaskTracker， JobTracker可以运行于集群中任一台计算机上。
- TaskTracker 负责执行任务，必须运行于DataNode上，即DataNode既是数据存储结点，也是计算结点。
- JobTracker将Map任务和Reduce任务分发给空闲的TaskTracker，让这些任务并行运行，并负责监控任务的运行情况。
- 如果某一个TaskTracker 出故障了，JobTracker会将其负责的任务转交给另一个空闲的TaskTracker重新运行。

分布式并行计算

- 下图是Hadoop Job Tracker示意图：



分布式并行计算

- 通常来说，MapRedcue作业是通过JobClient.rubJob(job)方法向master节点的JobTracker提交的，JobTracker接到JobClient的请求后把其加入作业队列中。JobTracker一直在等待JobClient通过RPC向其提交作业，而TaskTracker一直通过RPC向JobTracker发送心跳信号询问有没有任务可做，如果有，则请求JobTracker派发任务给它执行。
- 这是一个主动请求的任务，slave的TaskTracker主动向master的JobTracker请求任务。当TaskTracker接到任务后，通过自身调度在本slave建立起Task执行任务。

分布式并行计算

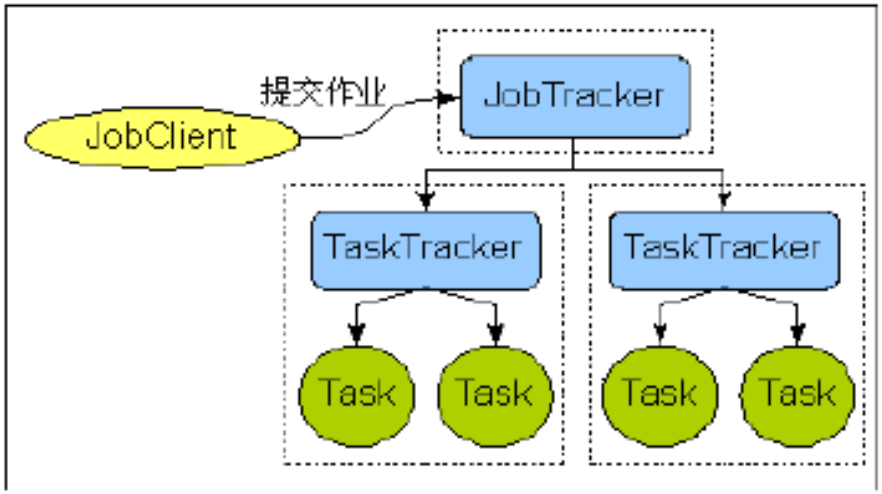
- MapReduce任务请求调度过程包括两个步骤：

（1）JobClient提交作业

JobClient向JobTracker提交作业。

（2）JobTacker调度作业

Task存于调度队列中，TaskTracker空闲时会请求任务，JobTracker通过调度算法取出一个task交给来请求的TaskTracker去执行。



本地计算

- 数据存储在哪一台计算机上，就由这台计算机进行这部分数据的计算，这样可以减少数据在网络上的传输，降低对网络带宽的需求。
- 在 Hadoop 这样的基于集群的分布式并行系统中，计算结点可以很方便地扩充，而因它所能够提供的计算能力近乎是无限的，但是由是数据需要在不同的计算机之间流动，故网络带宽变成了瓶颈，是非常宝贵的，“本地计算”是最有效的一种节约网络带宽的手段，业界把这形容为“移动计算比移动数据更经济”。

任务粒度

- 把原始大数据集切割成小数据集时，通常让小数据集小于或等于HDFS中一个Block的大小(缺省是64M)，这样能够保证一个小数据集位于一台计算机上，便于本地计算。
- 有 M 个小数据集待处理，就启动M个Map任务，注意这M个Map任务分布于N台计算机上并行运行，Reduce任务的数量R则可由用户指定。

Partition

- Partition是选择配置，主要作用是在多个Reduce的情况下，指定Map的结果由某一个Reduce处理，每一个Reduce都会有单独的输出文件。
- 把 Map 任务输出的中间结果按 key 的范围划分成 R 份(R 是预先定义的 Reduce 任务的个数)，划分时通常使用 hash 函数如: hash(key) mod R，这样可以保证某一段范围内的 key，一定是由一个 Reduce 任务来处理，可以简化 Reduce 的过程。

Combine

- 在partition之前，还可以对中间结果先做combine，即将中间结果中有相同 key的 <key, value> 对合并成一对。
- Combiner是可选择的，它的主要作用是在每一个Map执行完分析以后，在本地优先作Reduce的工作，减少在Reduce过程中的数据传输量。
- Combine的过程与Reduce的过程类似，很多情况下就可以直接使用Reduce函数，但combine是作为Map任务的一部分，在执行完 Map 函数后紧接着执行的。Combine能够减少中间结果中<key，value>对的数目，从而减少网络流量。

Reduce 任务取中间结果

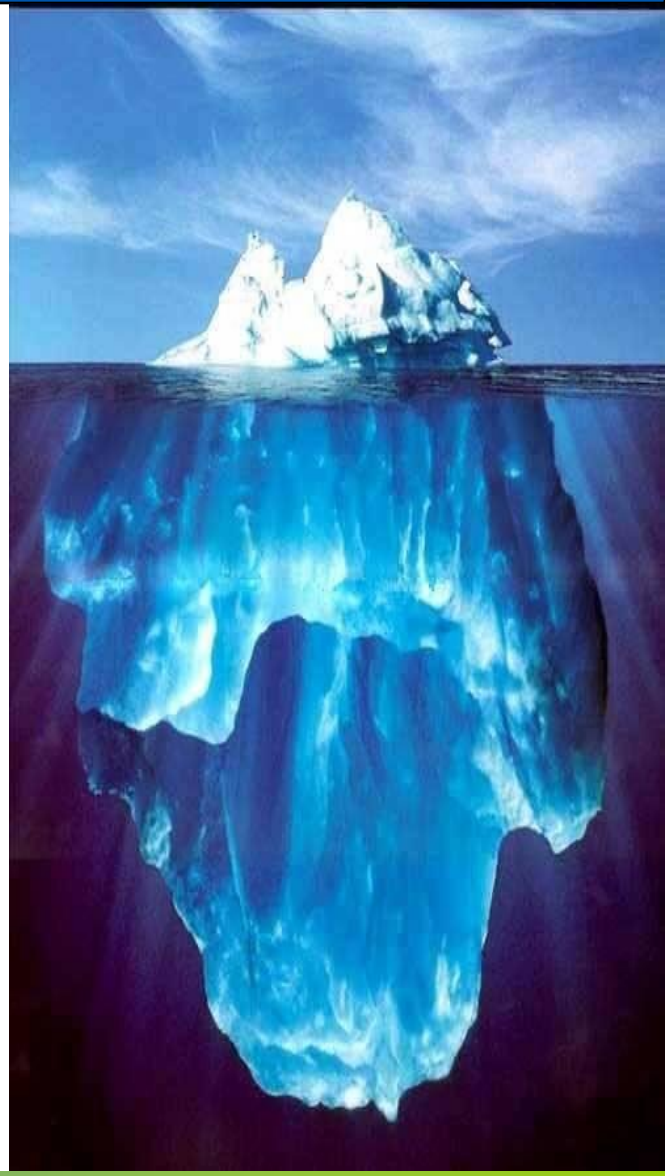
- Map 任务的中间结果在做完 Combine 和 Partition 之后，以文件形式存于本地磁盘。
- 中间结果文件的位置会通知主控 JobTracker， JobTracker 再通知 Reduce 任务到哪一个 DataNode 上去取中间结果。
- 注意所有的 Map 任务产生中间结果均按其 Key 用同一个 Hash 函数划分成了 R 份，R 个 Reduce 任务各自负责一段 Key 区间。每个 Reduce 需要向许多个 Map 任务结点取得落在其负责的 Key 区间内的中间结果，然后执行 Reduce 函数，形成一个最终的结果文件。

任务管道

- 有 R 个 Reduce 任务，就会有 R 个最终结果，很多情况下这 R 个最终结果并不需要合并成一个最终结果。因为这 R 个最终结果又可以做为另一个计算任务的输入，开始另一个并行计算任务。

课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



实例分析：WordCount

- 如果想统计过去10年计算机论文中出现次数最多的几个单词，看看大家都在研究些什么，可以大致采用以下几种方法：

1) 写一个小程序，把所有论文按顺序遍历一遍，统计每一个遇到的单词的出现次数，最后就可以知道哪几个单词最热门了（适合于数据集比较小，且非常有效的、实现最简单）。
2) 写一个多线程程序，并发遍历论文。理论上是可以高度并发的，因为统计一个文件时不会影响统计另一个文件。使用多核机器时比方法一高效。但是，写一个多线程程序要复杂得多。
3) 把作业交给多个计算机去完成。可以使用方法一的程序，部署到N台机器上去，然后把论文集分成N份，一台机器跑一个作业。这个方法跑得足够快，但是部署起来很麻烦，既要人工把论文集分开，复制到各台机器，还把N个运行结果进行整合。
4) 使用MapReduce！MapReduce本质上就是方法三，但如何拆分文件集，如何复制分发程序，如何整合结果这些都是框架定义好的。我们只要定义好这个任务（用户程序），其它都交给MapReduce。

WordCount设计思路

- 每个拿到原始数据的机器只要将输入数据切分成单词就可以了，所以可以在map阶段完成单词切分的任务。另外，相同单词的频数计算也可以并行化处理，可以将相同的单词交给一台机器来计算频数，然后输出最终结果，这个过程可以交给reduce阶段完成。至于将中间结果根据不同单词分组再发送给reduce机器，这正好是MapReduce过程中的shuffle能够完成的。

- 因此，WordCount的整个过程可表示为：

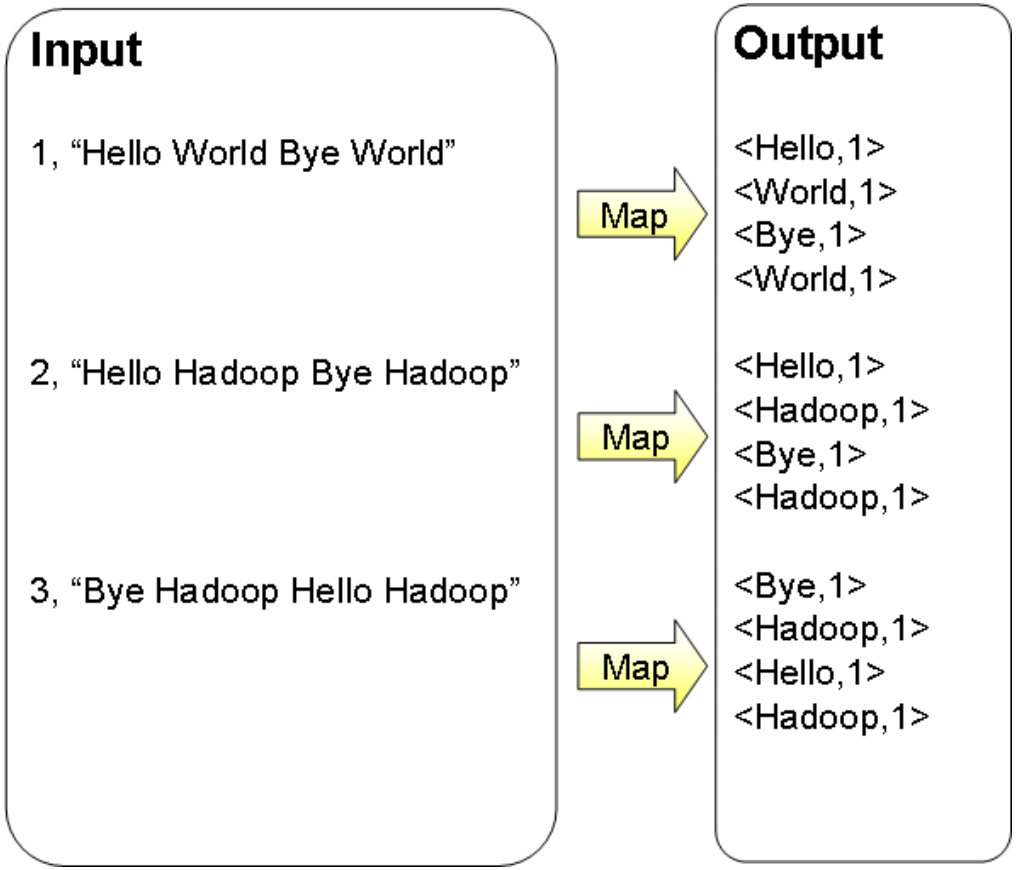
1. Map阶段完成由输入数据到单词切分的工作。
2. Shuffle阶段完成相同单词的聚集和分发工作（这个过程是MapReduce的默认过程，不用具体配置）。
3. Reduce阶段完成接收所有单词并计算其频数的工作。

WordCount设计思路

- MapReduce中传递的数据都是<key, value>形式的，并且shuffle排序聚集分发是按照key值进行的，所以，将map的输出设计成由word作为key，1作为value的形式，它表示单词出现了1次（map的输入采用Hadoop默认的输入方式，即文件的一行作为value，行号作为key）。
- Reduce的输入是map输出聚集后的结果，即<key, value-list>，具体到这个实例就是<word, {1,1,1,1,…}>，reduce的输出会设计成与map输出相同的形式，只是后面的数值不再是固定的1，而是具体算出的word所对应的频数。

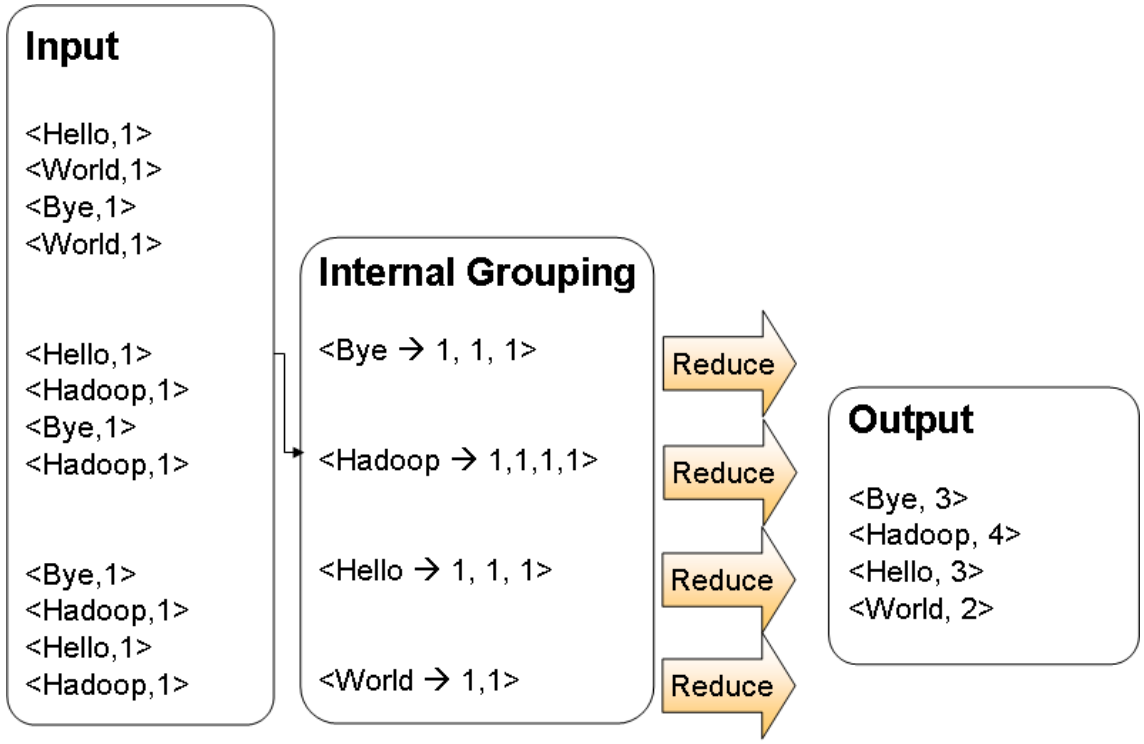
WordCount过程解释

- Map操作的输入是<key, value>形式，其中，key是文档中某行的行号，value是该行的内容。Map操作会将输入文档中每一个单词的出现输出到中间文件中去。



WordCount过程解释

- Reduce操作的输入是单词和出现次数的序列，如 <“Hello”， [1,1,1]> ， < “World”，[1,1] >， < “Bye”，[1,1,1] >， < “Hadoop”， [1,1,1,1] >等。然后根据每个单词，算出总的出现次数。
- 最后输出排序后的最终结果就会是：< “Bye”，3>，< “Hadoop”，4> ，< “Hello”，3>， < “World”，2>。



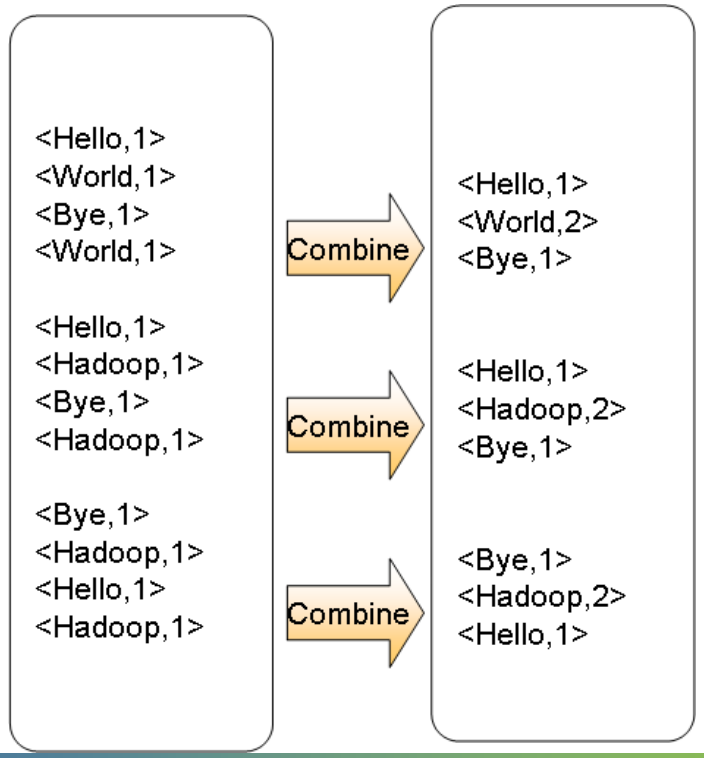
WordCount过程解释

- 整个MapReduce过程实际的执行顺序是：

1. MapReduce Library将Input分成M份。
2. Master将M份Job分给空闲状态的M个worker来处理；
3. 对于输入中的每一个<key， value> 进行Map操作，将中间结果缓冲在内存里。
4. 定期地（或者根据内存状态）将缓冲区中的中间信息刷写到本地磁盘上，并且把文件信息传回给Master（Master需要把这些信息发送给Reduce worker）。
5. R个Reduce worker开始工作，从不同的Map worker的Partition那里拿到数据，用key进行排序。
6. Reduce worker遍历中间数据，对每一个唯一Key，执行Reduce函数（参数是这个key以及相对应的一系列Value）。
7. 执行完毕后，唤醒用户程序，返回结果（最后应该有R份Output，每个Reduce Worker一个）。

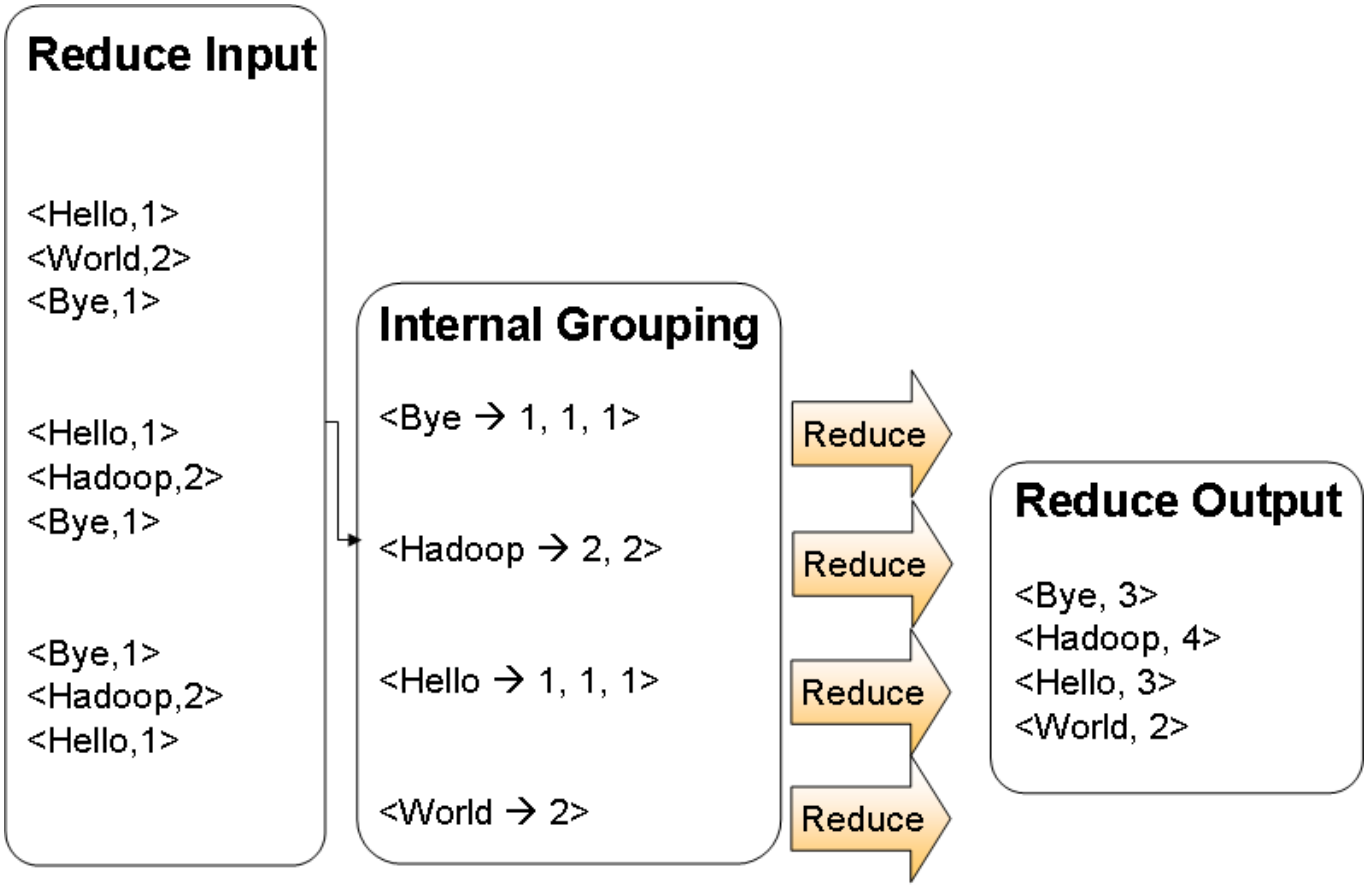
WordCount过程解释

- 对于上面的例子来说，每个文档中都可能会出现成千上万的 (“the”, 1)这样的中间结果，琐碎的中间文件必然导致传输上的损失。
- 因此，MapReduce还支持用户提供Combiner函数。这个函数通常与Reduce函数有相同的实现，不同点在于Reduce函数的输出是最终结果，而Combiner函数的输出是Reduce函数某一个输入的中间文件。



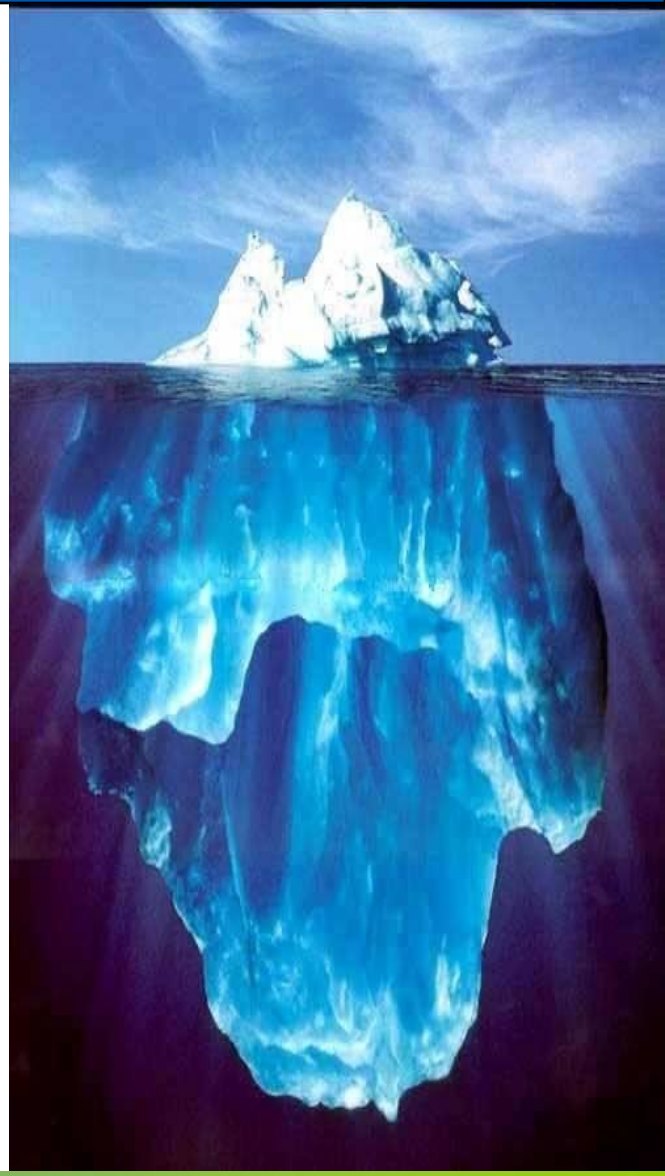
WordCount过程解释

- 下图是以Combine输出结果作为输入的Reduce过程示意图



课程提要

- 分布式并行编程
- MapReduce模型概述
- Map和Reduce函数
- MapReduce工作流程
- 并行计算的实现
- 实例分析：WordCount
- 新MapReduce框架Yarn



原MapReduce程序的流程及设计思路

- 首先用户程序 (JobClient) 提交了一个 job，job 的信息会发送到Job Tracker中，Job Tracker是MapReduce框架的中心，它需要与集群中的机器定时通信 (heartbeat), 需要管理哪些程序应该跑在哪些机器上，需要管理所有job失败、重启等操作。
- TaskTracker是MapReduce集群中每台机器都有的一个部分，它做的事情主要是监视自己所在机器的资源情况。
- TaskTracker同时监视当前机器的tasks运行状况。TaskTracker需要把这些信息通过heartbeat发送给JobTracker，JobTracker会搜集这些信息以确定新提交的job分配运行在哪些机器上。

原Hadoop MapReduce框架的问题

原来的 map-reduce 架构是简单明了的，在最初推出的几年，也得到了众多的成功案例，获得业界广泛的支持和肯定。但是，随着分布式系统集群的规模和其工作负荷的增长，原框架的问题逐渐浮出水面，主要的问题集中如下：

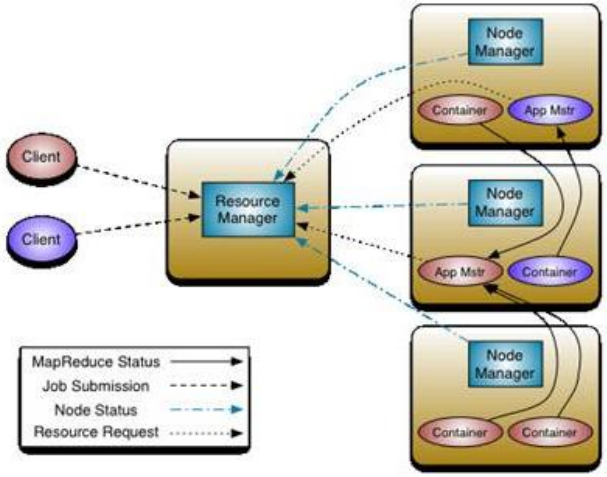
- JobTracker是Map-reduce的集中处理点，存在单点故障。
- JobTracker完成了太多的任务，造成了过多的资源消耗，当MapReduce job非常多的时候，会造成很大的内存开销，潜在来说，也增加了JobTracker失败的风险，这也是业界普遍总结出来的一个结论，即老Hadoop的MapReduce只能支持4000节点主机的上限。
- 在TaskTracker端，以MapReduce任务的数目作为资源的表示过于简单，没有考虑到CPU和内存的占用情况，如果两个大内存消耗的任务被调度到了一块，很容易出现OOM（Out of Memory）。

原Hadoop MapReduce框架的问题

- 在TaskTracker端，把资源强制划分为map task slot和reduce task slot, 如果当系统中只有map task或者只有reduce task的时候，会造成资源的浪费，也就是前面提过的集群资源利用的问题。
- 源代码层面分析的时候，会发现代码非常难读，常常因为一个类（class）做了太多的事情，代码量达3000多行，造成类的任务不清晰，增加bug修复和版本维护的难度。
- 从操作的角度来看，现在的Hadoop MapReduce框架在有任何重要的或者不重要的变化时(例如bug修复，性能提升和特性化)，都会强制进行系统级别的升级更新。更糟的是，它不管用户的喜好，强制让分布式集群系统的每一个用户端同时更新。这些更新会让用户为了验证他们之前的应用程序是不是适用新的Hadoop版本而浪费大量时间。

新Hadoop Yarn框架原理及运作机制

- **ResourceManager**：ResourceManager是一个中心的服务，它做的事情是调度、启动每一个Job所属的ApplicationMaster，并且监控ApplicationMaster的存在情况。细心的读者会发现：Job里面所在的task的监控、重启等等内容不见了。这就是 ApplicationMaster存在的原因。ResourceManager负责作业与资源的调度。接收JobSubmitter提交的作业，按照作业的上下文(Context) 信息以及从NodeManager收集来的状态信息，启动调度过程，分配一个Container作为ApplicationMaster。
- **NodeManager**：功能比较专一，就是负责Container状态的维护，并向ResourceManager保持心跳。
- **ApplicationMaster**：负责一个Job生命周期内的所有工作，类似老的框架中 JobTracker。但是要注意，每一个 Job（不是每一种）都有一个 ApplicationMaster，它可以运行在ResourceManager以外的机器上。



Yarn框架相对于老的MapReduce框架的优势

- 这个设计大大减小了JobTracker（也就是现在的ResourceManager）的资源消耗，并且让监测每一个Job子任务 (tasks) 状态的程序分布式化了，更安全、更优雅。
- 在新的Yarn中，ApplicationMaster是一个可变更的部分，用户可以对不同的编程模型写自己的ApplicationMaster，让更多类型的编程模型能够跑在Hadoop集群中，可以参考Hadoop Yarn官方配置模板中的mapred-site.xml配置。
- 对于资源的表示以内存为单位 (在目前版本的Yarn 中，没有考虑CPU的占用)，比之前以剩余slot数目更合理。
- 老的框架中，JobTracker一个很大的负担就是监控job下的tasks的运行状况，现在，这个部分就扔给ApplicationMaster做了，而ResourceManager中有一个模块叫做ApplicationsMasters(注意不是ApplicationMaster)，它是监测ApplicationMaster 的运行状况，如果出问题，会将其在其他机器上重启。

主讲教师和助教

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

助教：赖明星

单位：厦门大学计算机科学系数据库实验室2011级硕士研究生（导师：林子雨）
E-mail: mingxinglai@gmail.com
个人主页：http://mingxinglai.com

欢迎访问《大数据技术基础》2013班级网站：http://dblab.xmu.edu.cn/node/423
本讲义PPT存在配套教材《大数据技术基础》，请到上面网站下载。

Department of Computer Science, Xiamen University, September, 2013