

# 分布式计算

## MapReduce介绍

Weixiong Rao 饶卫雄  
Tongji University 同济大学软件学院  
2023 秋季  
[wxrao@tongji.edu.cn](mailto:wxrao@tongji.edu.cn)

# 目录

- 概述

- ◆ 举例: 人口普查



- MapReduce 体系结构

- ◆ 数据流

- ◆ 执行流

- ◆ 容错

# 全国人口普查

- 假定有10,000员工，其职责就是分发调查表确定每个城市的居住人口
- 如何完成这些任务？

全国人口普查的“数据流”

United States Census 2010

This is the official form for all the people at this address. It is quick and easy, and your answers are protected by law.

U.S. DEPARTMENT OF COMMERCE  
Economics and Statistics Administration  
U.S. CENSUS BUREAU

Use a blue or black pen.  
**Start here**

The Census must count every person living in the United States on April 1, 2010.  
Before you answer Question 1, count the people living in this house, apartment, or mobile home using our guidelines.

- Count all people, including babies, who live and sleep here most of the time.

The Census Bureau also conducts counts in institutions and other places, so:

- Do not count anyone living away either at college or in the Armed Forces.
- Do not count anyone in a nursing home, jail, prison, detention facility, etc., on April 1, 2010.
- Leave these people off your form, even if they will return to live here after they leave college, the nursing home, the military, jail, etc. Otherwise, they may be counted twice.

The Census must also include people without a permanent place to stay, so:

- If someone who has no permanent place to stay is staying here on April 1, 2010, count that person. Otherwise, he or she may be missed in the census.

1. How many people were living or staying in this house, apartment, or mobile home on April 1, 2010?

Number of people =

2. Were there any additional people staying here April 1, 2010 that you did not include in Question 1? Mark ☒ all that apply.

- ☐ Children, such as newborn babies or foster children
- ☐ Relatives, such as adult children, cousins, or in-laws
- ☐ Nonrelatives, such as roommates or live-in baby sitters
- ☐ People staying here temporarily
- ☐ No additional people

3. Is this house, apartment, or mobile home — Mark ☒ ONE box.

- ☐ Owned by you or someone in this household with a mortgage or loan? Include home equity loans.
- ☐ Owned by you or someone in this household free and clear (without a mortgage or loan)?
- ☐ Rented?
- ☐ Occupied without payment of rent?

4. What is your telephone number? We may call if we don't understand an answer.

Area Code + Number  -  -

OMB No. 0607-0919-C: Approval Expires 12/31/2011.  
Form D-61 (1-16-2009)

5. Please provide information for each person living here. Start with a person living here who owns or rents this house, apartment, or mobile home. If the owner or renter lives somewhere else, start with any adult living here. This will be Person 1.  
What is Person 1's name? Print name below.

Last Name  MI

First Name

6. What is Person 1's sex? Mark ☒ ONE box.  
☐ Male ☐ Female

7. What is Person 1's age and what is Person 1's date of birth? Please report babies as age 0 when the child is less than 1 year old. Print numbers in boxes.

Age on April 1, 2010  Month  Day  Year of birth

→ NOTE: Please answer BOTH Question 8 about Hispanic origin and Question 9 about race. For this census, Hispanic origins are not races.

8. Is Person 1 of Hispanic, Latino, or Spanish origin?

- ☐ No, not of Hispanic, Latino, or Spanish origin
- ☐ Yes, Mexican, Mexican Am., Chicano
- ☐ Yes, Puerto Rican
- ☐ Yes, Cuban
- ☐ Yes, another Hispanic, Latino, or Spanish origin — Print origin, for example, Argentinean, Colombian, Dominican, Nicaraguan, Salvadoran, Spaniard, and so on.

9. What is Person 1's race? Mark ☒ one or more boxes.

- ☐ White
- ☐ Black, African Am., or Negro
- ☐ American Indian or Alaska Native — Print name of enrolled or principal tribe.
- ☐ Asian Indian
- ☐ Japanese
- ☐ Native Hawaiian
- ☐ Chinese
- ☐ Korean
- ☐ Guamanian or Chamorro
- ☐ Filipino
- ☐ Vietnamese
- ☐ Samoan
- ☐ Other Asian — Print race, for example, Hmong, Laotian, Thai, Pakistani, Cambodian, and so on.
- ☐ Other Pacific Islander — Print race, for example, Fijian, Tongan, and so on.
- ☐ Some other race — Print race.

10. Does Person 1 sometimes live or stay somewhere else?

- ☐ No
- ☐ Yes — Mark ☒ all that apply.
  - ☐ In college housing
  - ☐ In the military
  - ☐ At a seasonal or second residence
  - ☐ For child custody
  - ☐ In jail or prison
  - ☐ In a nursing home
  - ☐ For another reason

→ If more people were counted in Question 1, continue with Person 2.

USCENSUSBUREAU



# 有点复杂

- 大家会休假、生病，工作节奏不一样
- 有些表格填写出错、甚至给扔丢了
- 如果上级领导休假，表格给谁？
- 如何管控整个普查这个事情的进度？
- ...

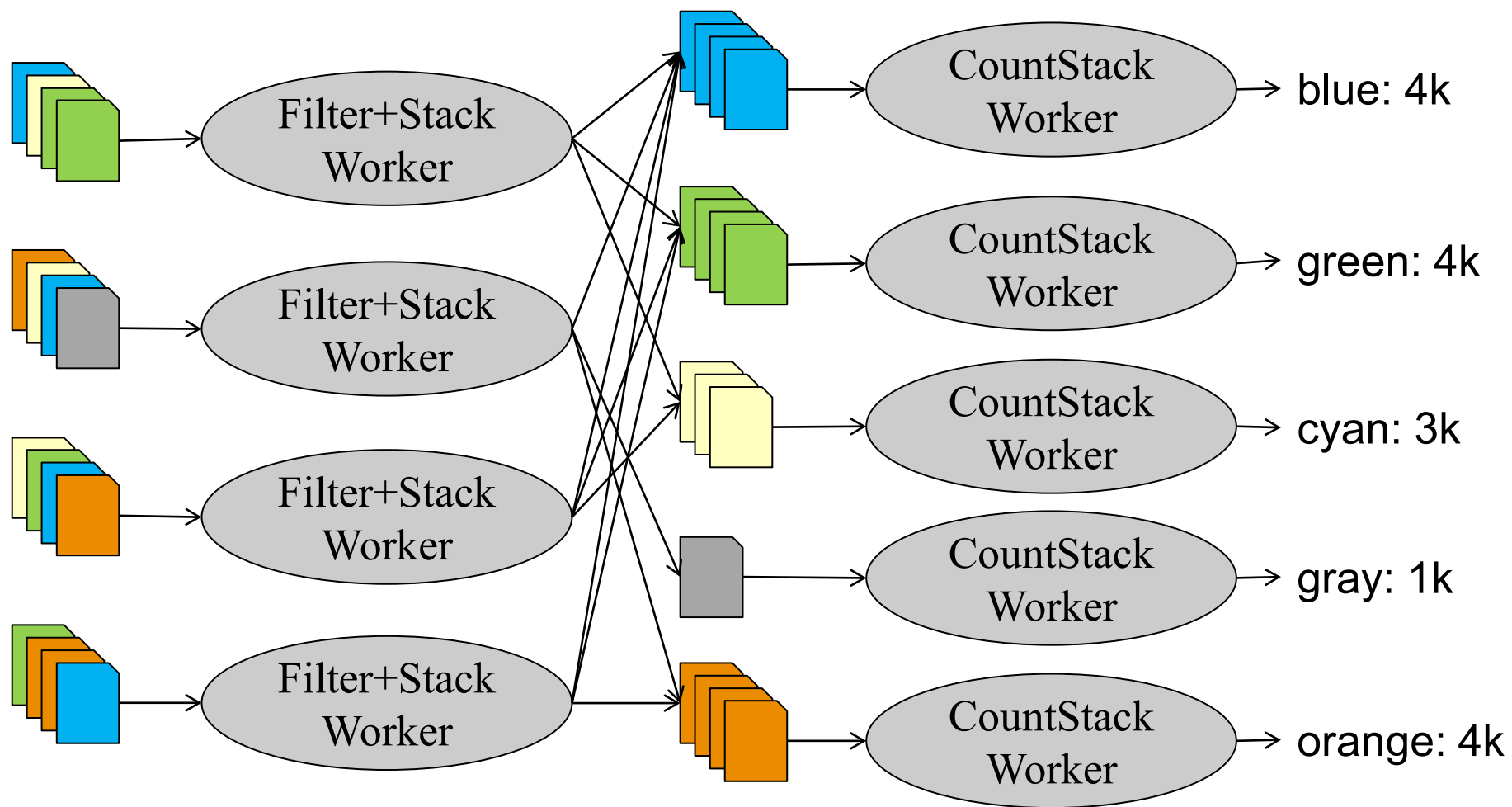
# 来点思考把

- 最大的困难是什么？
  - ◆ 人口普查表填写工作复杂吗？
  - ◆ 如果不复杂, 为什么这个事情这么困难呢？
- 答案有弹性吗？
- 如何在所有的员工之间平衡工作量？
  - ◆ 哪些影响因素？公平性
- 如何泛化推广到其他场景？
  - ◆ 人口普查 → 全国经济总量、全球人口分布...

# 说实话,没有人愿意每天应付这些繁琐事情!!!

- 如果有一个系统可以自动的帮助你处理这些细节,生活不是很美好吗?
- 理想情况下,我只想告诉这个系统,我需要做的事情就可以啦,后面的处理细节让这个系统处理就好!
- 那个系统就是**MapReduce**计算框架

# 抽象成数据流



# 再来一次抽象

## ■ 两类workers:

- ◆ 处理输入数据，输出生成结果“stacks”
- ◆ 处理结果“stacks”，然后aggregate生成最终结果

## ■ MapReduce中的两个重要部分:

- ◆ map: 处理 (item\_key, value), 生成一个或者多个 (stack\_key, value') 对
- ◆ reduce: 处理 (stack\_key, {set of value'}), 生成一个或者多个输出结果 — (stack\_key, agg\_value)

以示区分item\_key，称之为reduce key



# 为什么要用MapReduce?

- 考虑如下场景:

- ◆ 有一堆数据, e.g., Google公司最近3年的所有搜索记录数据
- ◆ 任务: 处理该数据, 求得那个输入的关键字最热?

- 怎么做?

- 和之前的人口普查例子类比下:

- ◆ 计算本身不怎么复杂, 但是如何并行化和分布式处理, 当然还有容错, 并非易事

- 思路: 设计一个编程框架或者语言!

- ◆ 编写一个简单的程序来表达 (尽管很简易) 计算, 然后让该框架的执行环境来处理最难的部分(即: 并行化和分布式处理, 及容错)

# 目录

- 概述



- ◆ Census example

- MapReduce 体系结构

- ◆ 数据流

- ◆ 执行流

- ◆ 容错



# MapReduce是什么？

- 可以说，MR差不多是最出名的分布式编程模型
- Google数据分析的基石
  - ◆ 构建在其之上的编程语言：Sawzall,  
<http://labs.google.com/papers/sawzall.html>
  - ◆ ... Sawzall has become one of the most widely used programming languages at Google. ... [O]n one dedicated Workqueue cluster with 1500 Xeon CPUs, there were 32,580 Sawzall jobs launched, using an average of 220 machines each. While running those jobs, 18,636 failures occurred (application failure, network outage, system crash, etc.) that triggered rerunning some portion of the job. The jobs read a total of  $3.2 \times 10^{15}$  bytes of data (2.8PB) and wrote  $9.9 \times 10^{12}$  bytes (9.3TB).
  - ◆ 其他类似的编程语言：Yahoo's Pig Latin and Pig; Microsoft's Dryad
- 开源项目：Hadoop  
<http://hadoop.apache.org/>

# MapReduce编程模型

- 分布式功能编程原语
- Lisp原语:
  - ◆ map (该函数应用到一个集合厘米的所有数据项)
  - ◆ reduce (该函数应用到具有相同key的所有数据项)
- MapReduce:
  - ◆ 用户自定义数据map, 应用到所有输入数据,  
map: (key,value)  $\rightarrow$  (key, value)
  - ◆ 另外一个用户自定义函数  
reduce: (key, {set of values})  $\rightarrow$  result
  - ◆ n个计算节点, 每个节点处理一部分数据
- 所有的节点运行map函数以处理所负责的数据, 生成新数据结果(包含keys)
  - ◆ 这些生成数据按照对应的key集合起来, 然后shuffled洗牌, 最终应用reduced函数
  - ◆ Dataflow就穿过 GFS上的临时文件

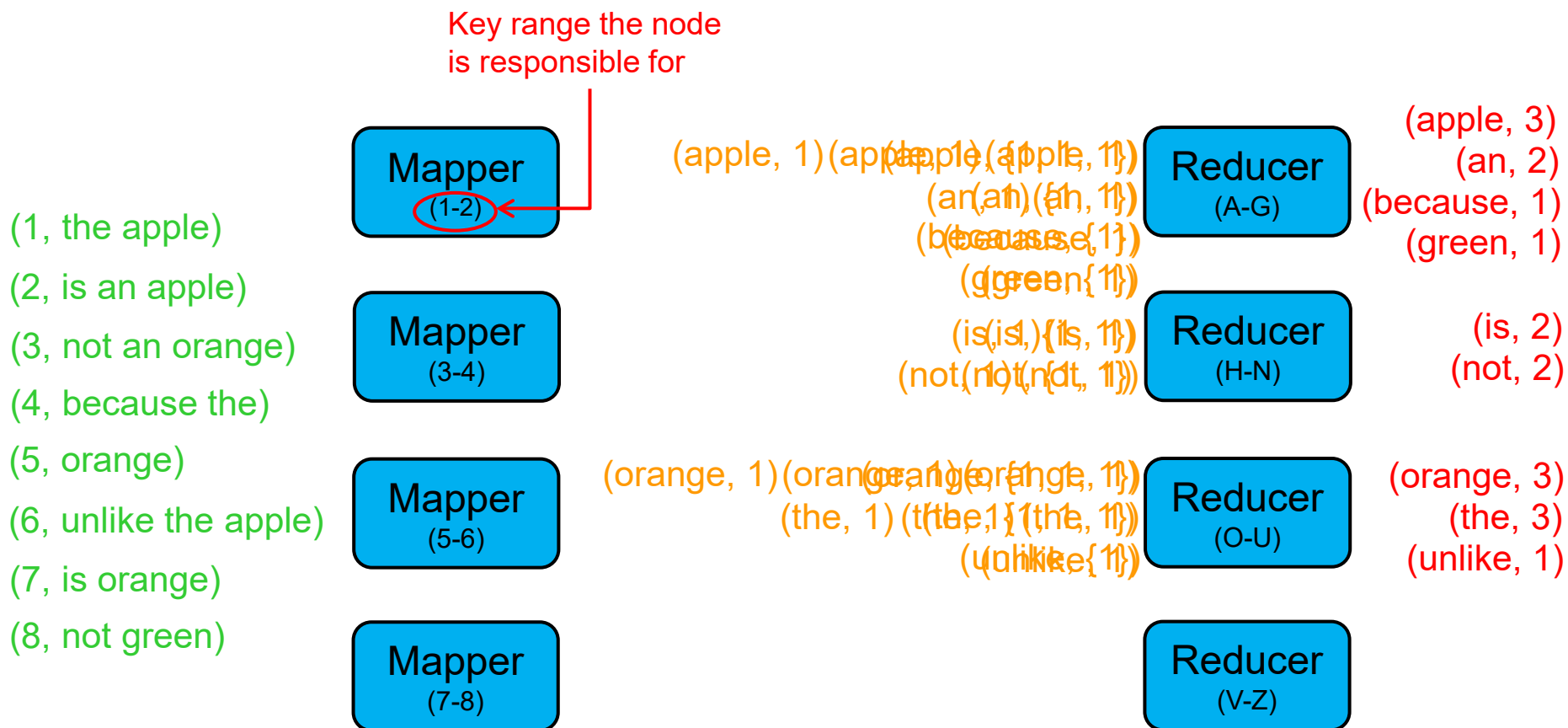
# 简单例子: Word count

```
map(String key, String value) {  
    // key: document name, line no  
    // value: contents of line  
    for each word w in value:  
        emit(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(key, result)  
}
```

- 任务: 给一个文档集合, 计数每个单词的频次
  - ◆ 输入: Key-value对 (document:lineNumber, text)
  - ◆ 输出: Key-value对 (word, #occurrences)
  - ◆ 中间的临时key-value对是怎样的呢?

# 简单例子: Word count



① 每个 mapper 收到一些KV对作为输入

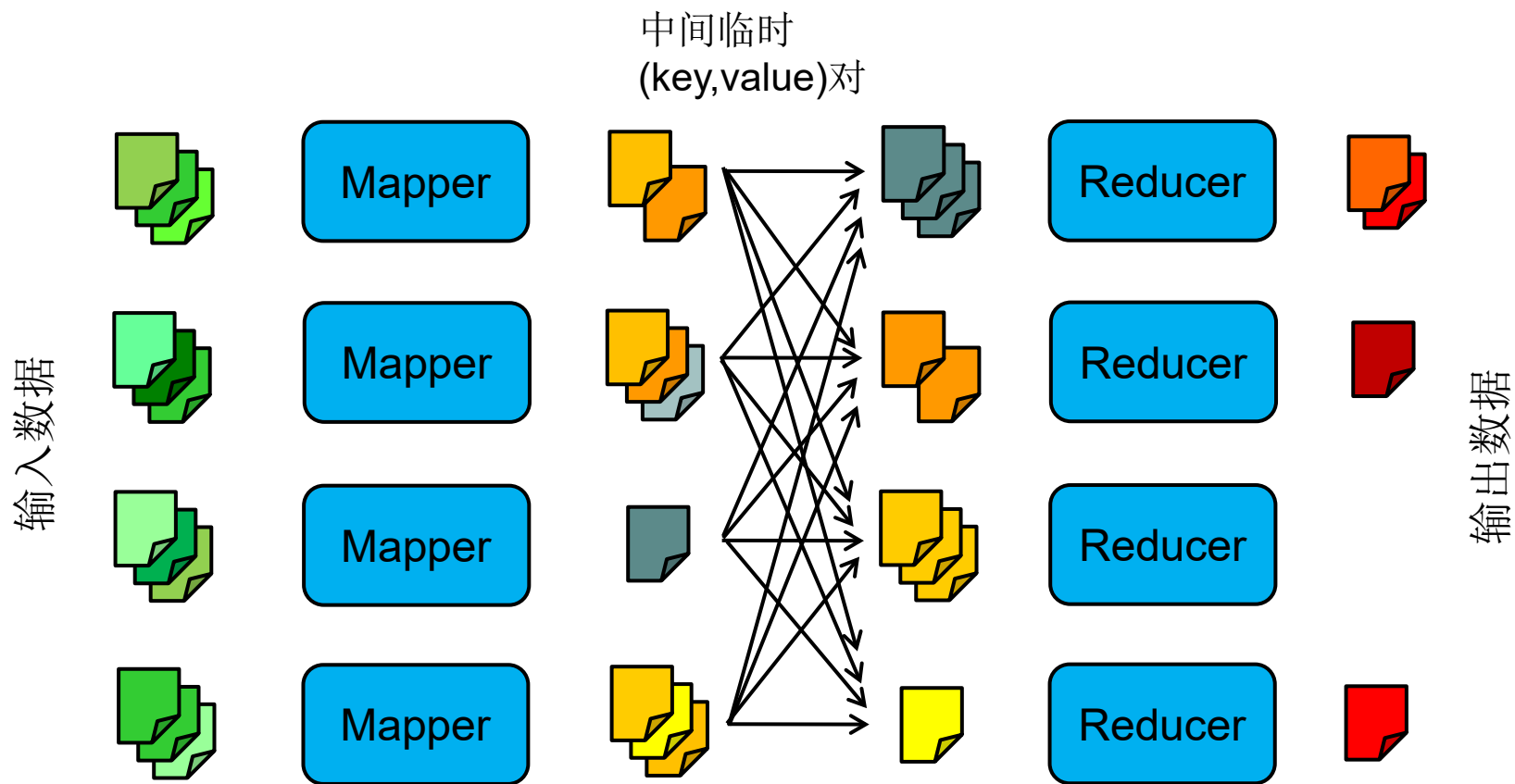
② Mappers 逐一处理这些KV对

③ Mapper输出的KV对发送至所负责的reducer

④ Reducers 对所接收的KV对按照key进行排序并进行分组

⑤ reducers按照一次一个组的方式处理KV对

# MapReduce dataflow



“Shuffle”洗牌

‘dataflow’怎么理解？  
为什么可扩展性这么强大？

# 其他例子

- 分布式 **grep** – 匹配一个给定模式pattern的所有行
  - ◆ Map: 按照该pattern进行过滤
  - ◆ Reduce: 输出结果
- 计数URL的访问频率
  - ◆ Map: 输出: 每个URL作为key, val是计数值1
  - ◆ Reduce: 汇总求和每个URL的所有计数值
- Web链接图的求逆
  - ◆ Map: 当有从source到target的链接, 输出(target,source)对
  - ◆ Reduce: 并联操作, 输出 (target,list(source))
- 到排表
  - ◆ Map: 输出 (word,documentID)
  - ◆ Reduce: 合并生成 (word,list(documentID))



# MR编程陷阱

## ■ Mapper 及 reducer 应该是无状态stateless

- ◆ map + reduce 返回值避免使用static变量, 别指望通过static变量传送数据处理结果!
- ◆ 理由: 不清楚哪个<key-value对>会在哪个workers上进行处理!

```
HashMap h = new HashMap();  
map(key, value) {  
    if (h.contains(key)) {  
        h.add(key, value);  
        emit(key, "X");  
    }  
}
```

Wrong!

## ■ 不要写自己的I/O!

- ◆ 不要写从文件系统进行读写文件的代码
- ◆ MapReduce框架会自动进行 I/O的操作:
  - ▣ 所有输入数据会自动作为map和reduce这两个函数的参数
  - ▣ 任何自定义函数的输出结果均应该通过emit完成

```
map(key, value) {  
    File foo =  
        new File("xyz.txt");  
    while (true) {  
        s = foo.readLine();  
        ...  
    }  
}
```

Wrong!

# MR编程陷阱

```
map(key, value) {  
    emit("FOO", key + " " + value);  
}
```

Wrong!

```
reduce(key, value[]) {  
    /* do some computation on  
    all the values */  
}
```

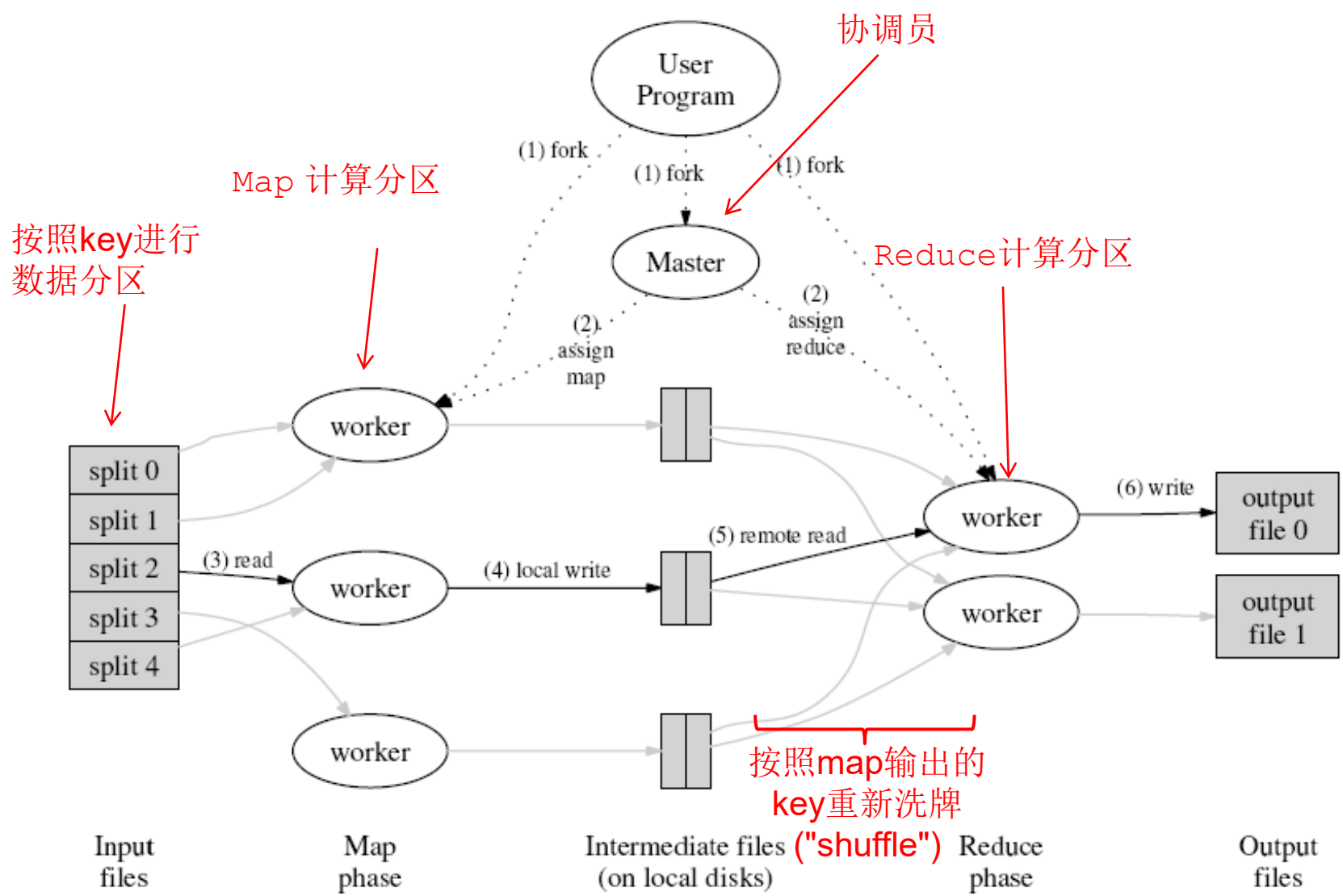
- Mapper避免一个key对应过多的数据(vals)
  - ◆ 特别是不要把所有的数据对应到一个key!!
  - ◆ 否则某一个reducer工作负载过高，而其他reducers空闲!
  - ◆ 可接受情况: 某些reducers的工作负载比其他reducers负载要高
    - 举例: WordCount, 负责‘and’的reducer负载会比负责‘syzygy’的reducer负载要高!

# 设计MapReduce算法

- 关键思考点: 哪些地方应该由map来做, 哪些由reduce做?
  - ◆ map 只会处理每个收到的<key-value>对, 不会理睬其他<key-value>对
    - 举例: 过滤不需要的<key-value>对
  - ◆ map 可释放多个中间临时<key-value>对, 尽管输入一个<key-value>对
    - 举例: 输入文本数据, map 生成每个单词对应的(word,1)对
  - ◆ reduce 汇总数据; 对一个中间临时key, 可收到的中间临时values
    - 举例: WordCount中reduce可以收到多个(word,1)对, 这样一个word对应多个1
- 中间临时<key-val>对的格式
  - ◆ 如果reduce需要将多个vals进行捆绑一起处理, map就必须对这些vals使用同样的keys!

2022/12/13

# MapReduce data flow细节



# 其他细节

- 如何使得MapReduce框架动起来？
- 文件系统file system (分布到所有计算节点):
  - ◆ 存储输入数据、输出数据和临时中间结果
- 驱动程序driver program (在某一个节点上运行):
  - ◆ 指定在哪里找到输入和输出结果
  - ◆ 指定哪段代码是mapper和reducers函数
  - ◆ 还可以定制运行处理的过程
- 运行环境runtime system (控制节点):
  - ◆ 监督整个任务的执行处理
  - ◆ JobTracker

# 其他细节

## ■ 计算分区Vs数据分区

- ◆ 所有的数据都是通过分布式文件系统进行访问操作
- ◆ 计算节点产生数据是按照key的方式完成(便于汇总合并)
- ◆ Master程序负责计算的调度、使得每个计算节点保持忙的状态
- ◆ Master程序指定有多个数据分区、哪些数据分区已经完成处理
  - 原子性的提交到磁盘

## ■ Locality: Master程序尽量在有数据的节点上安排计算任务

- ◆ 就近原则
- ◆ 避免数据的搬运，消化网络开销，增加延迟

## ■ Master对那些托后腿的计算节点 (比较慢的节点)

- ◆ 重新分配该节点上的任务到其他(快)的计算节点

# 如果一个计算节点宕机了

- 靠分布式文件系统(分布在所有计算节点)
- 2种类型的 (宕机)故障:
  - ◆ 节点在完成写数据之后宕机
    - 文件系统(很有可能)有该数据的副本
  - ◆ 节点在完成写数据之前宕机
    - JobTracker就会发现该job没有往前进，在其他的计算节点重新启动该job
- (当然啦，实际工作的计算节点就会少一个了...)
- 但是，如何master节点宕机了，咋办？

# 其他的一些情况

## ■ Locality就近原则

- ◆ 尽量在有数据的计算节点上安排map任务

## ■ 任务粒度

- ◆ 分配几个mapper? 几个reducer?

## ■ 处理托后腿的计算

- ◆ 启动备用的任务

## ■ 节省网络带宽

- ◆ E.g., 启动combiners函数



# MapReduce可扩展规模

## ■ Google的论文提及:

- ◆ ... Sawzall has become one of the most widely used programming languages at Google. ...  
[O]n one dedicated Workqueue cluster with 1500 Xeon CPUs, there were 32,580 Sawzall jobs launched, using an average of 220 machines each.  
While running those jobs, 18,636 failures occurred (application failure, network outage, system crash, etc.) that triggered rerunning some portion of the job. The jobs read a total of  $3.2 \times 10^{15}$  bytes of data (2.8PB) and wrote  $9.9 \times 10^{12}$  bytes (9.3TB).

## ■ 我们还会讲MapReduce开发的语言 Pig Latin, HiveQL

# 敬请期待



下节课:  
继续MapReduce编程

# 分布式计算

MapReduce编程

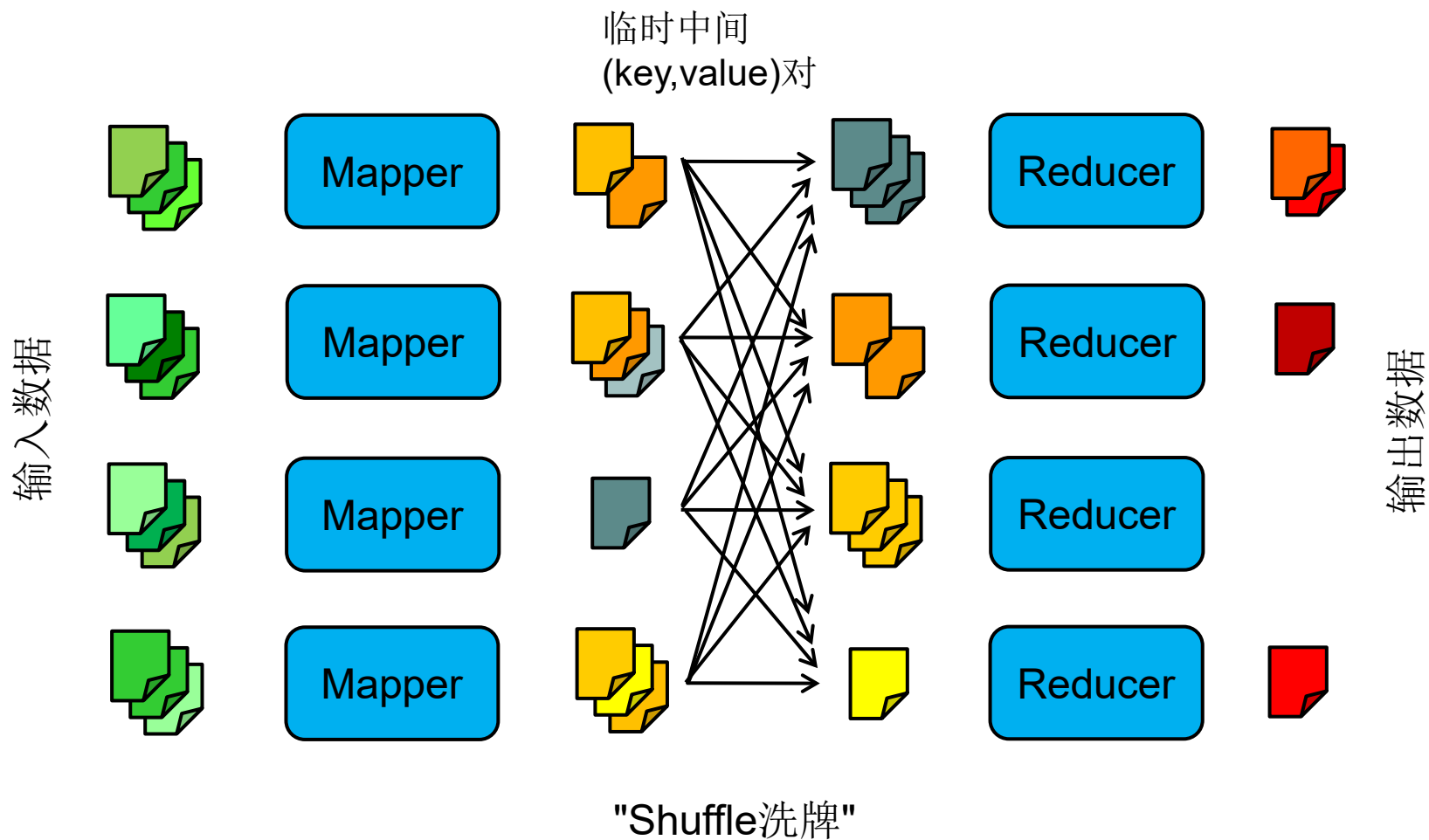
Weixiong Rao 饶卫雄

Tongji University 同济大学软件学院

2023 秋季

wxrao@tongji.edu.cn

# 回顾: MapReduce dataflow



# 回顾: MapReduce

```
map(key:URL, value:Document)
{
```

生成中间的<key-value>  
对, 随后发送至reducer

```
    String[] words = value.split(" ");
    for each w in words
        emit(w, 1);
}
```

```
reduce(rkey:String, rvalues:Integer[])
{
```

```
    Integer result = 0;
    foreach v in rvalues
        result = result + v;
    emit(rkey, result);
}
```

数据类型取决于输入数据

reduce的 types 通常与map的types不同

reduce收到该rkey  
值得全部中间值

map() 和 reduce() 均是无状态:  
避免使用跨多个调用函数之间还  
保留的全局变量

Reducer通过emit()释放出来的  
kv对都会发送至最终的output

# 今天课堂内容

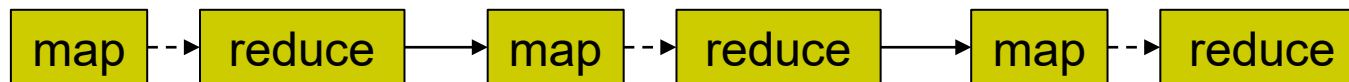
## ■ MapReduce的单趟算法

- ◆ Filtering 算法
- ◆ Aggregation 算法
- ◆ Intersections交集 & joins连接
- ◆ Partial Cartesian products部分笛卡儿乘法
- ◆ Sorting



# 基本概念

- 单趟算法single-pass
- 设计一个MR算法，需要分解为filter / collect / aggregate几个步骤
  - ◆ Filter/collect是 **map**函数的一部分
  - ◆ Collect/aggregate 则是**reduce**函数的一部分
- 注意：某些算法需要多个map / reduce 阶段 – maps & reduces函数链



# Filtering 算法

- 目的: 按照给定的特征模式, 查找文件行/文件/表纪录
- 举例:
  - ◆ 通过grep命令过滤查找所有发送到\*.tongji.edu.cn/\*的Web日志记录
  - ◆ 查找Web日志中的所有192.168.2.1访问的主机名
  - ◆ 定位所有包括 'Apple' 及 'Jobs' 的文件
- 总体来说: map函数作多数的计算, 而 reduce 则几乎没有做实际计算



# Aggregation 算法

- 任务: 计算一个数据集合的最大值、求和值、平均值 ...
- 举例:
  - ◆ 统计所有发送到 \*. tongji.edu.cn/\* 的请求总和
  - ◆ 查找最火的域名
  - ◆ 求每个Web网站每个页面的访问平均
- **map** 不做实际计算, **reduce** 完成汇总计算

# 难点的例子

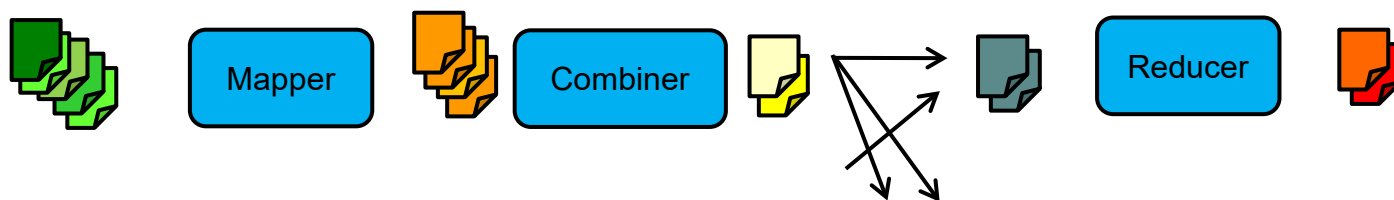
- 任务：设计一个Amazon CloudFront 这样CDN的计费帐单
  - ◆ 输入：edge servers的日志文件. 每个domain域名对应两个文件：
    - access\_log-www.foo.com-20111006.txt: HTTP 访问日志
    - ssl\_access\_log-www.foo.com-20111006.txt: HTTPS 访问日志
    - 示例文件行：

```
158.130.53.72 - - [06/Oct/2011:16:30:38 -0400] "GET /largeFile.ISO HTTP/1.1" 200 8130928734 "-" "Mozilla/5.0 (compatible; MSIE 5.01; Win2000)"
```
    - Mapper处理 (filename,line) 的纪录
  - ◆ 计费策略：
    - 考虑到数据次数和数据流量两个因素
    - 10,000个HTTP请求计费\$0.0075
    - 10,000 HTTPS请求计费\$0.0100
    - 1 GB 数据流量计费 \$0.12
  - ◆ 输出结果格式：(domain, grandTotal) 清单

# 复杂的汇总函数: Combiners

- 一些函数可以分解为多个步骤:

- ◆ 可以把两个子分区的计数进一步汇总, 得到整个分区的计数结果
- ◆ 可以把两个子分区的max值进行比较, 得到整个分区的唯一max值



- 同一台机器上的多个map 任务有可能是发送到同一个reduce key之上

举例:

- ◆ `map(1, "Apple juice") -> ("apple", 1), ("juice", 1)`
- ◆ `map(2, "Apple sauce") -> ("apple", 1), ("sauce", 1)`
- ◆ `combiner: ("apple", [1, 1]) -> ("apple", 2)`

# 交集 Intersection & 连接 joins

- 任务: 对多个输入内容求同样值的交集
- 举例:
  - ◆ 通过一个倒排表结构, 找同时包括“data”和“centric”的所有文件
  - ◆ 查找教授任课表和学生上课表中相同课程的  
<professor,student>所有偶对

# 细说Joins

## ■ 两种实现方法:

- ◆ Reduce-side join: 基本和 intersection一样
  - 把所有源表纪录中join属性相同值(“join key”)发送到同一个reducer
  - 在reducer中比较接收到偶对中的内容值
  - 如果符合join等值, 则emit 这些内容值
- ◆ Map-side join: 需要join 表均采用同一种方式进行分区
  - 需要在map函数直接访问文件
  - 较为复杂

## ■ 那种方式更加有效呢?

- ◆ Lin & Dyer, Chapter 3

# 局部笛卡儿乘法

- 任务: 查找特点复杂关系, 比如根据任意两组数据之间的距离
- 举例:
  - ◆ 查找相互距离100米之内的一对地址组合
- 比较难以实现并行化
  - ◆ 不过可以将整个数据集分区, 或者连接到一些兰大marks
  - ◆ 如何生成landmarks?(使用clustering?)

# 排序

- 任务: 对输入数据进行排序
- 举例:
  - ◆ 返回Google索引所覆盖的所有域名及每个域名所包括的页面数, 并按照页面数对所有域名进行排序
- MR编程模型本身支持这种任务, 但是我们可以通过代码实现可以完成
  - ◆ Shuffle!

# 今天课堂内容

## ■ MapReduce的单趟算法

◆ Filtering 算法



◆ Aggregation 算法



◆ Intersections交集 & joins连接



◆ Partial Cartesian products部分笛卡儿乘法



◆ **Sorting**



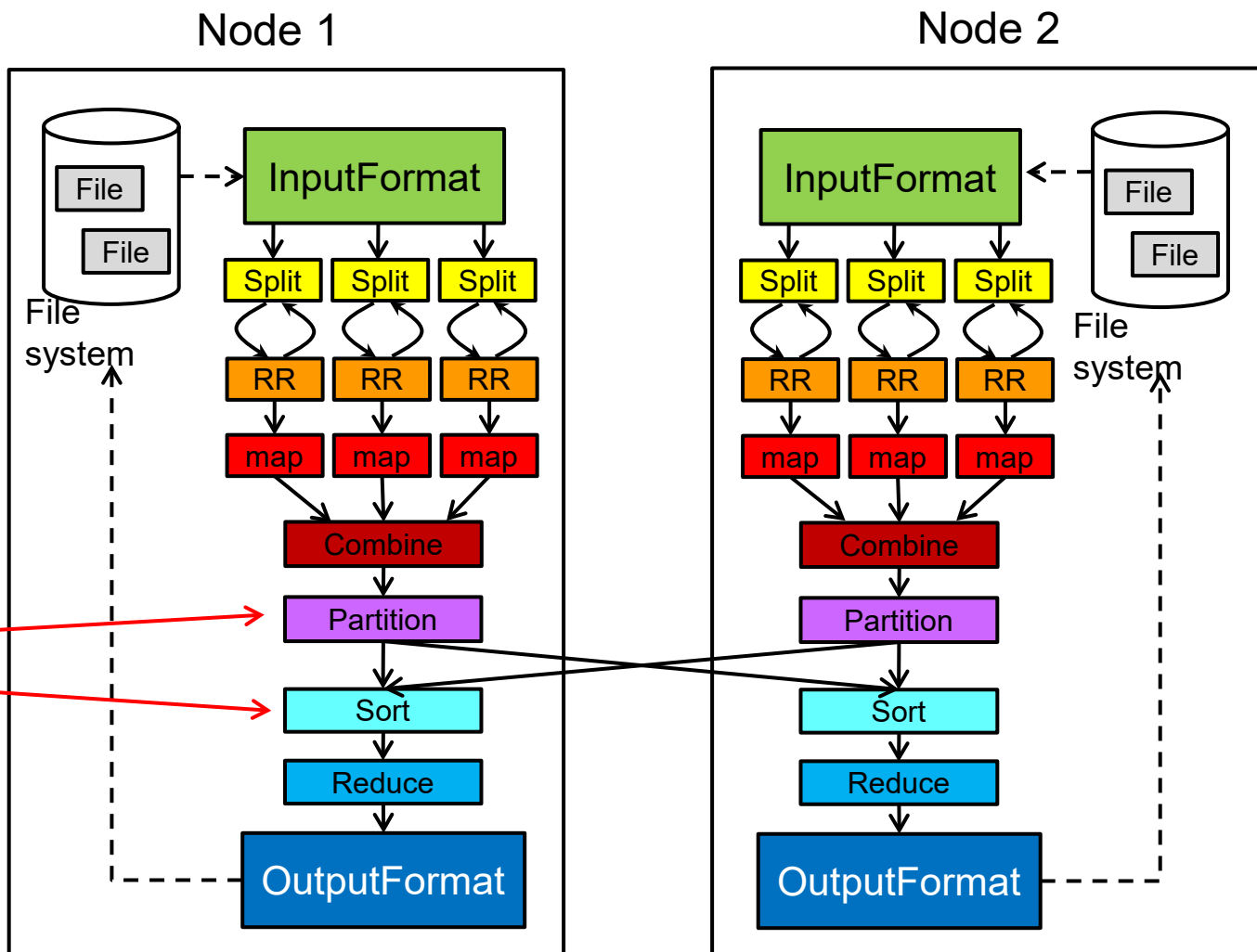
<http://codingjunkie.net/secondary-sort/>



# Shuffle

Shuffle 本质上包括如下2个步骤:

- Partition
- Sort



# Shuffle可以当成一种排序的手段

- 可以利用Shuffle阶段当中的每节点的排序运算
  - ◆ 如果只有一个reducer, 可以直接得到排序结果
  - ◆ 如果有多个reducers, 至少可以得到部分排序结果
    - ▣ 最终通过单趟mr程序, 合并所有部分排序结果生成最终排序结果
- 举例
  - ◆ 返回Google索引所覆盖的所有域名及每个域名所包括的页面数, 并按照页面数对所有域名进行排序

# MR的优缺点

- MR擅长解决的问题？
  - ◆ ... 是否可以通过一趟MR完成？
  - ◆ ... 是否需要多趟MR完成？
- 是否有算法不能通过MR有效解决的？
- 还有那些问题MR根本就不能完成的？
- 是否还有其他的方法可以做海量大数据分析的？
  - ◆ MapReduce是否总是唯一最快、最有效的方法？

# 小结: MapReduce 算法

- 相当多的计算任务可以通过MR程序来表达实现
  - ◆ Filtering & aggregation + 及其各种组合
  - ◆ Joins
  - ◆ 可以通过多趟MR程序或者固定次数的迭代
- 不擅长的计算任务
  - ◆ 局部笛卡尔乘法
    - 虽然不怎么有效，但是还是能工作
  - ◆ 排序算法
    - 本质上MR不能完成，但是可以取巧的通过shuffle完成



下节课  
**Hadoop**

# 分布式计算

## 12-Hadoop

Weixiong Rao 饶卫雄

Tongji University 同济大学软件学院

2023 秋季

wxrao@tongji.edu.cn

# 本节课的内容

- A brief history of Hadoop
- Writing jobs for Hadoop
  - ◆ Mappers, reducers, drivers
  - ◆ Compiling and running a job
- Hadoop Distributed File System (HDFS)
  - ◆ Node types; read and write operation
  - ◆ Accessing data in HDFS
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ...
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution



# 2002-2004: Lucene and Nutch

## ■ Early 2000s: Doug Cutting develops two open-source search projects:

- ◆ Lucene: Search indexer
  - ▣ Used e.g., by Wikipedia
- ◆ Nutch: A spider/crawler (with Mike Carafella)



## ■ Nutch

- ◆ Goal: Web-scale, crawler-based search
- ◆ Written by a few part-time developers
- ◆ Distributed, 'by necessity'
- ◆ Demonstrated 100M web pages on 4 nodes, but true 'web scale' still very distant



# 2004-2006: GFS and MapReduce

## ■ 2003/04: GFS, MapReduce papers published

- ◆ Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: "The Google File System", SOSP 2003
- ◆ Jeffrey Dean and Sanjay Ghemawat: "MapReduce: Simplified Data Processing on Large Clusters", OSDI 2004
- ◆ Directly addressed Nutch's scaling issues

## ■ GFS & MapReduce added to Nutch

- ◆ Two part-time developers over two years (2004-2006)
- ◆ Crawler & indexer ported in two weeks
- ◆ Ran on 20 nodes
- ◆ Much easier to program and run, scales to several 100M web pages, but still far from web scale

# 2006-2008: Yahoo

## ■ 2006: Yahoo hires Cutting

- ◆ Provides engineers, clusters, users, ...
- ◆ Big boost for the project; Yahoo spends tens of M\$
- ◆ Not without a price: Yahoo has a slightly different focus (e.g., security) than the rest of the project; delays result

## ■ Hadoop project split out of Nutch

- ◆ Finally hit web scale in early 2008



## ■ Cutting is now at Cloudera

- ◆ Startup; started by three top engineers from Google, Facebook, Yahoo, and a former executive from Oracle
- ◆ Has its own version of Hadoop; software remains free, but company sells support and consulting services
- ◆ Was elected chairman of Apache Software Foundation

# Who uses Hadoop?

- Hadoop is running search on some of the Internet's largest sites:
  - ◆ Amazon Web Services: Elastic MapReduce
  - ◆ AOL: Variety of uses, e.g., behavioral analysis & targeting
  - ◆ EBay: Search optimization
  - ◆ Facebook: Reporting/analytics, machine learning
  - ◆ Fox Interactive Media: MySpace, Photobucket, Rotten T.
  - ◆ Last.fm: Track statistics and charts
  - ◆ IBM: Blue Cloud Computing Clusters
  - ◆ LinkedIn: People You May Know
  - ◆ Rackspace: Log processing
  - ◆ Twitter: Store + process tweets, log files, other data
  - ◆ Yahoo: >40,000 nodes; biggest cluster is 4,500 nodes/455PB

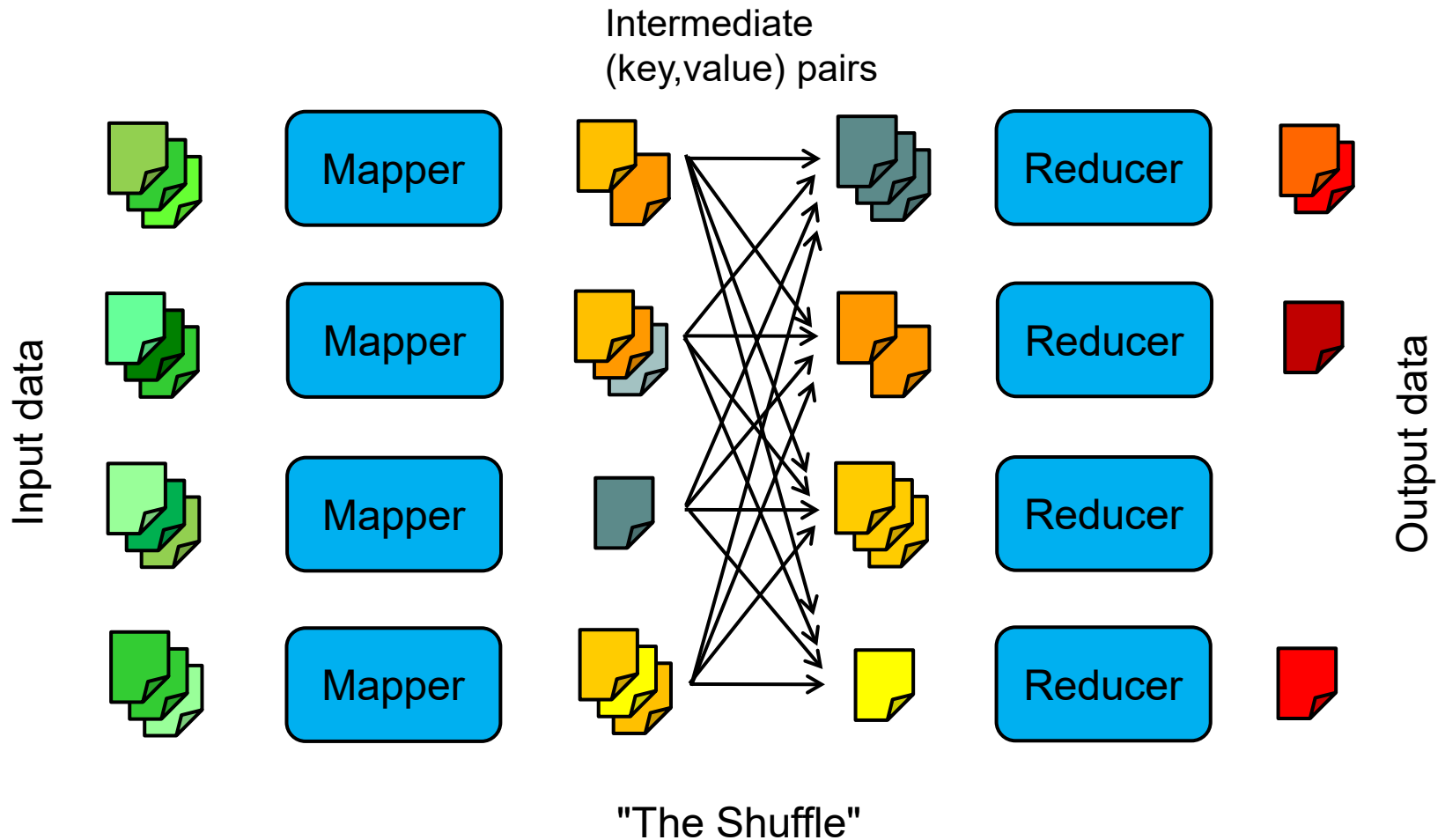
# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job
- Hadoop Distributed File System (HDFS)
  - ◆ Node types; read and write operation
  - ◆ Accessing data in HDFS
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ...
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution

# Simplified scenario

- In this section, I will demonstrate how to use Hadoop in standalone mode
  - ◆ Useful for development and debugging (NOT for production)
  - ◆ Single node (e.g., your laptop computer)
  - ◆ No jobtrackers or tasktrackers
  - ◆ Data in local file system, not in HDFS
- This is how the Hadoop installation in your virtual machine works
- Later: Fully-distributed mode
  - ◆ Used when running Hadoop on actual clusters

# Recap: MapReduce dataflow



# What do we need to write?

- A mapper
  - ◆ Accepts (key,value) pairs from the input
  - ◆ Produces intermediate (key,value) pairs, which are then shuffled
- A reducer
  - ◆ Accepts intermediate (key,value) pairs
  - ◆ Produces final (key,value) pairs for the output
- A driver
  - ◆ Specifies which inputs to use, where to put the outputs
  - ◆ Chooses the mapper and the reducer to use
- Hadoop takes care of the rest!!
  - ◆ Default behaviors can be customized by the driver

# Hadoop data types

- Hadoop uses its own serialization
  - ◆ Java serialization is known to be very inefficient
- Result: A set of special data types
  - ◆ All implement the 'Writable' interface
  - ◆ Most common types shown above; also has some more specialized types (SortedMapWritable, ObjectWritable, ...)
  - ◆ **Caution:** Behavior somewhat unusual

Name	Description	JDK equivalent
IntWritable	32-bit integers	Integer
LongWritable	64-bit integers	Long
DoubleWritable	Floating-point numbers	Double
Text	Strings	String



# The Mapper

Input format  
(file offset, line)

Intermediate format  
can be freely chosen

```
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.io.*;

public class FooMapper extends Mapper<LongWritable, Text, Text, Text> {
    public void map(LongWritable key, Text value, Context context) {
        context.write(new Text("foo"), value);
    }
}
```

- Extends abstract 'Mapper' class
  - ◆ Input/output types are specified as type parameters
- Implements a 'map' function
  - ◆ Accepts (key,value) pair of the specified type
  - ◆ Writes output pairs by calling 'write' method on context
  - ◆ Mixing up the types will cause problems at runtime (!)

# The Reducer

```
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.io.*;

public class FooReducer extends Reducer<Text, Text, IntWritable, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context)
        throws java.io.IOException, InterruptedException
    {
        for (Text value: values)
            context.write(new IntWritable(4711), value);
    }
}
```

Intermediate format  
(same as mapper output)

Output format

Note: We may get multiple values for the same key!

- Extends abstract 'Reducer' class
  - ◆ Must specify types again (must be compatible with mapper!)
- Implements a 'reduce' function
  - ◆ Values are passed in as an 'Iterable'
  - ◆ **Caution:** These are NOT normal Java classes. Do not store them in collections - content can change between iterations!

# The Driver

```
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class FooDriver {
    public static void main(String[] args) throws Exception {
        Job job = new Job();
        job.setJarByClass(FooDriver.class);

        FileInputFormat.addInputPath(job, new Path("in"));
        FileOutputFormat.setOutputPath(job, new Path("out"));

        job.setMapperClass(FooMapper.class);
        job.setReducerClass(FooReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```





Mapper&Reducer are  
in the same Jar as  
FooDriver

Input and Output  
paths

Format of the (key,value)  
pairs output by the  
reducer

- Specifies how the job is to be executed
  - ◆ Input and output directories; mapper & reducer classes

# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop 
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job 
- Hadoop Distributed File System (HDFS)
  - ◆ Node types; read and write operation
  - ◆ Accessing data in HDFS
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ...
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution

# Manual compilation

- **Goal:** Produce a JAR file that contains the classes for mapper, reducer, and driver
  - ◆ This can be submitted to the Job Tracker, or run directly through Hadoop

- **Step #1:** Put hadoop-core-1.0.3.jar into classpath:

```
export CLASSPATH=$CLASSPATH:/path/to/hadoop/hadoop-core-1.0.3.jar
```

- **Step #2:** Compile mapper, reducer, driver:

```
javac FooMapper.java FooReducer.java FooDriver.java
```

- **Step #3:** Package into a JAR file:

```
jar cvf Foo.jar *.class
```

- **Alternative:** "Export..." / "Java JAR file" in Eclipse

# Compilation with Ant

```
<project name="foo" default="jar" basedir="."/>
  <target name="init">
    <mkdir dir="classes"/>
  </target>

  <target name="compile" depends="init">
    <javac srcdir="src" destdir="classes" includes="*.java" debug="true"/>
  </target>

  <target name="jar" depends="compile">
    <jar destfile="foo.jar">
      <fileset dir="classes" includes="**/*.class"/>
    </jar>
  </target>

  <target name="clean">
    <delete dir="classes"/>
    <delete file="foo.jar"/>
  </target>
</project>
```

Directory where  
source files  
are kept

Makes the JAR  
file

Clean up any  
derived files

- Apache Ant: A build tool for Java (~"make")
  - ◆ Run "ant jar" to build the JAR automatically
  - ◆ Run "ant clean" to clean up derived files (like make clean)

# Standalone mode installation

## ■ What is standalone mode?

- ◆ Installation on a single node
- ◆ No daemons running (no Task Tracker, Job Tracker)
- ◆ Hadoop runs as an 'ordinary' Java program
- ◆ Used for debugging

## ■ How to install Hadoop in standalone mode?

- ◆ See Textbook Appendix A
- ◆ Already done in your VM image

# Running a job in standalone mode

- **Step #1: Create & populate input directory**
  - ◆ Configured in the Driver via `addInputPath()`
  - ◆ Put input file(s) into this directory (ok to have more than 1)
  - ◆ Output directory must not exist yet
- **Step #2: Run Hadoop**
  - ◆ As simple as this: `hadoop jar <jarName> <driverClassName>`
  - ◆ Example: `hadoop jar foo.jar upenn.nets212.FooDriver`
  - ◆ In verbose mode, Hadoop will print statistics while running
- **Step #3: Collect output files**








# Recap: Writing simple jobs for Hadoop

- Write a mapper, reducer, driver
  - ◆ Custom serialization → Must use special data types (Writable)
  - ◆ Explicitly declare all three (key,value) types
- Package into a JAR file
  - ◆ Must contain class files for mapper, reducer, driver
  - ◆ Create manually (javac/jar) or automatically (ant)
- Running in standalone mode
  - ◆ `hadoop jar foo.jar FooDriver`
  - ◆ Input and output directories in local file system
- More details: Chapters 2,5,6 of your textbook

# Wait a second...

- Wasn't Hadoop supposed to be very scalable?
  - ◆ Work on Petabytes of data, run on thousands of machines
- Some more puzzle pieces are needed
  - ◆ Special file system that can a) hold huge amounts of data, and b) feed them into MapReduce efficiently
    - Hadoop Distributed File System (HDFS)
  - ◆ Framework for distributing map and reduce tasks across many nodes, coordination, fault tolerance...
    - Fully distributed mode
  - ◆ Mechanism for customizing dataflow for particular applications (e.g., non-textual input format, special sort...)
    - Hadoop data flow

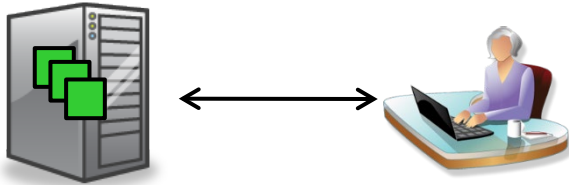
# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop 
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job 
- Hadoop Distributed File System (HDFS) 
  - ◆ Node types; read and write operation
  - ◆ Accessing data in HDFS
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ...
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution

# What is HDFS?

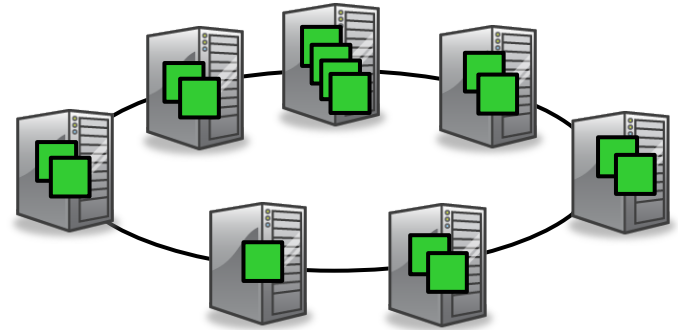
- HDFS is a distributed file system
  - ◆ Makes some unique tradeoffs that are good for MapReduce
- What HDFS does well:
  - ◆ Very large read-only or append-only files (individual files may contain Gigabytes/Terabytes of data)
  - ◆ Sequential access patterns
- What HDFS does not do well:
  - ◆ Storing lots of small files
  - ◆ Low-latency access
  - ◆ Multiple writers
  - ◆ Writing to arbitrary offsets in the file

# HDFS versus NFS



Network File System (NFS)

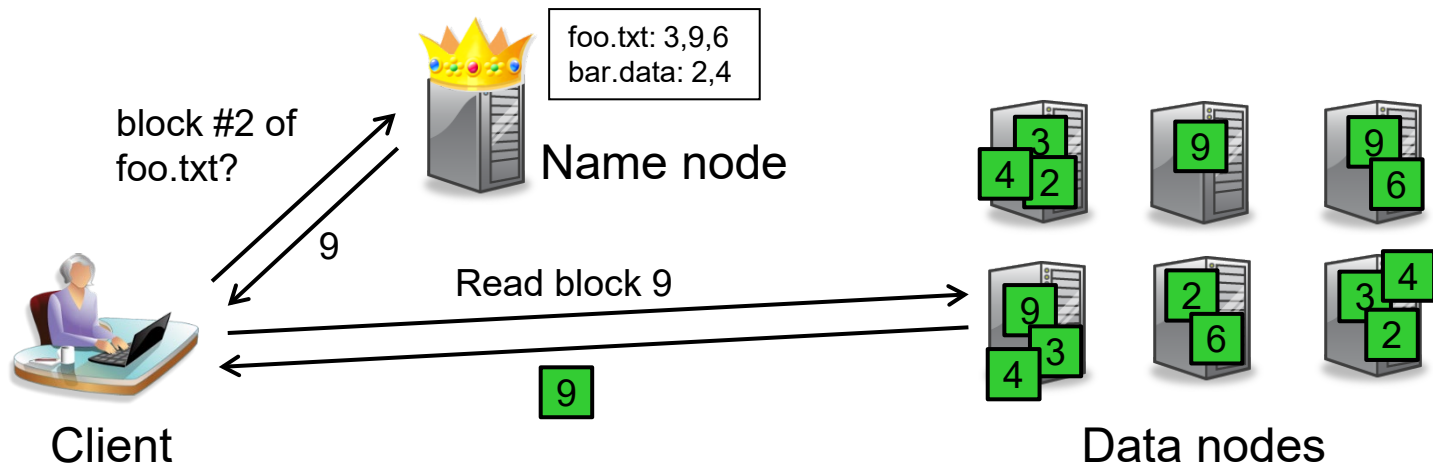
- Single machine makes part of its file system available to other machines
- Sequential or random access
- **PRO:** Simplicity, generality, transparency
- **CON:** Storage capacity and throughput limited by single server



Hadoop Distributed File System (HDFS)

- Single virtual file system spread over many machines
- Optimized for sequential read and local accesses
- **PRO:** High throughput, high capacity
- **"CON":** Specialized for particular types of applications

# How data is stored in HDFS



- **Files are stored as sets of (large) blocks**
  - ◆ Default block size: 64 MB (ext4 default is 4kB!)
  - ◆ Blocks are replicated for durability and availability
  - ◆ What are the advantages of this design?
- **Namespace is managed by a single name node**
  - ◆ Actual data transfer is directly between client & data node
  - ◆ Pros and cons of this decision?

# The Namenode



Name node

```
foo.txt: 3,9,6  
bar.data: 2,4  
blah.txt: 17,18,19,20  
xyz.img: 8,5,1,11
```

fsimage

```
Created abc.txt  
Appended block 21 to blah.txt  
Deleted foo.txt  
Appended block 22 to blah.txt  
Appended block 23 to xyz.img  
...
```

edits

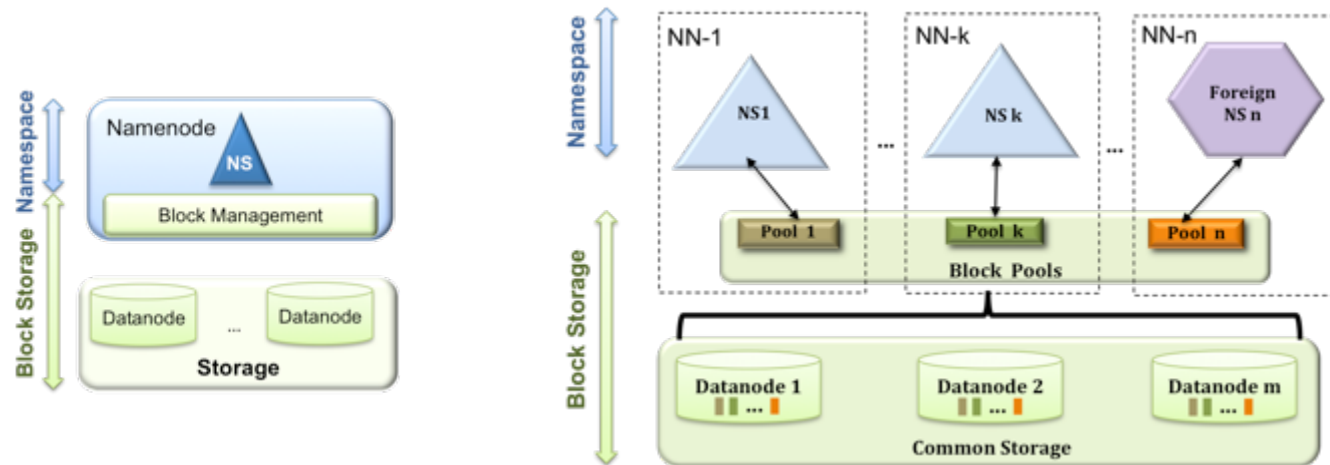
- State stored in two files: fsimage and edits
  - ◆ fsimage: Snapshot of file system metadata
  - ◆ edits: Changes since last snapshot
- Normal operation:
  - ◆ When namenode starts, it reads fsimage and then applies all the changes from edits sequentially
  - ◆ Pros and cons of this design?

# The Secondary Namenode

- What if the state of the namenode is lost?
  - ◆ Data in the file system can no longer be read!
- Solution #1: Metadata backups
  - ◆ Namenode can write its metadata to a local disk, and/or to a remote NFS mount
- Solution #2: Secondary Namenode
  - ◆ Purpose: Periodically merge the edit log with the fsimage to prevent the log from growing too large
  - ◆ Has a copy of the metadata, which can be used to reconstruct the state of the namenode
  - ◆ But: State lags behind somewhat, so data loss is likely if the namenode fails



# HDFS Federation










Classical HDFS

Federated HDFS

- **Newer HDFS versions support federation**
  - ◆ Multiple independent Namenodes share a common storage
  - ◆ Each Namenode has a separate **block pool**, so it can generate block IDs without coordinating w/other Namenodes

# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop 
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job 
- Hadoop Distributed File System (HDFS) 
  - ◆ Node types; read and write operation 
  - ◆ Accessing data in HDFS
- Hadoop internals 
  - ◆ Dataflow: Input format, partitioner, combiner, ...
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution

# Accessing data in HDFS

```
[ahae@carbon ~]$ ls -la /tmp/hadoop-ahae/dfs/data/current/
total 209588
drwxrwxr-x 2 ahae ahae      4096 2015-10-06 15:46 .
drwxrwxr-x 5 ahae ahae      4096 2015-10-06 15:39 ..
-rw-rw-r-- 1 ahae ahae 11568995 2015-10-06 15:44 blk_-3562426239750716067
-rw-rw-r-- 1 ahae ahae   90391 2015-10-06 15:44 blk_-3562426239750716067_1020.meta
-rw-rw-r-- 1 ahae ahae      4 2015-10-06 15:40 blk_5467088600876920840
-rw-rw-r-- 1 ahae ahae     11 2015-10-06 15:40 blk_5467088600876920840_1019.meta
-rw-rw-r-- 1 ahae ahae 67108864 2015-10-06 15:44 blk_7080460240917416109
-rw-rw-r-- 1 ahae ahae   524295 2015-10-06 15:44 blk_7080460240917416109_1020.meta
-rw-rw-r-- 1 ahae ahae 67108864 2015-10-06 15:44 blk_-8388309644856805769
-rw-rw-r-- 1 ahae ahae   524295 2015-10-06 15:44 blk_-8388309644856805769_1020.meta
-rw-rw-r-- 1 ahae ahae 67108864 2015-10-06 15:44 blk_-9220415087134372383
-rw-rw-r-- 1 ahae ahae   524295 2015-10-06 15:44 blk_-9220415087134372383_1020.meta
-rw-rw-r-- 1 ahae ahae     158 2015-10-06 15:40 VERSION
[ahae@carbon ~]$
```

- HDFS implements a separate namespace
  - ◆ Files in HDFS are not visible in the normal file system
  - ◆ Only the blocks and the block metadata are visible
  - ◆ HDFS cannot be (easily) mounted
    - Some FUSE drivers have been implemented for it (and a NFS proxy)

# Accessing data in HDFS

```
[ahae@carbon ~]$ /usr/local/hadoop/bin/hadoop fs -ls /user/ahae
Found 4 items
-rw-r--r--  1 ahae supergroup      1366 2015-10-06 15:46 /user/ahae/README.txt
-rw-r--r--  1 ahae supergroup        0 2015-10-06 15:35 /user/ahae/input
-rw-r--r--  1 ahae supergroup        0 2015-10-06 15:39 /user/ahae/input2
-rw-r--r--  1 ahae supergroup 212895587 2015-10-06 15:44 /user/ahae/input3
[ahae@carbon ~]$
```

- File access is through the hadoop command
- Examples:
  - ◆ `hadoop fs -put [file] [hdfsPath]` Stores a file in HDFS
  - ◆ `hadoop fs -ls [hdfsPath]` List a directory
  - ◆ `hadoop fs -get [hdfsPath] [file]` Retrieves a file from HDFS
  - ◆ `hadoop fs -rm [hdfsPath]` Deletes a file in HDFS
  - ◆ `hadoop fs -mkdir [hdfsPath]` Makes a directory in HDFS

# Alternatives to the command line

- Getting data in and out of HDFS through the command-line interface is a bit cumbersome
- Alternatives have been developed:
  - ◆ FUSE file system: Allows HDFS to be mounted under Unix
  - ◆ NFS proxy: Makes HDFS look like a file server
  - ◆ WebDAV share: Can be mounted as filesystem on many OSes
  - ◆ HTTP: Read access through namenode's embedded web svr
  - ◆ FTP: Standard FTP interface
  - ◆ ...

# Accessing HDFS directly from Java

- Programs can read/write HDFS files directly
  - ◆ Not needed in MapReduce; I/O is handled by the framework
- Files are represented as URIs
  - ◆ Example: `hdfs://localhost/user/ahae/example.txt`
- Access is via the `FileSystem` API
  - ◆ To get access to the file: `FileSystem.get()`
  - ◆ For reading, call `open()` -- returns `InputStream`
  - ◆ For writing, call `create()` -- returns `OutputStream`

# What about permissions?

- Since 0.16.1, Hadoop has rudimentary support for POSIX-style permissions
  - ◆ rwx for users, groups, 'other' -- just like in Unix
  - ◆ 'hadoop fs' has support for chmod, chgrp, chown
- But: POSIX model is not a very good fit
  - ◆ Many combinations are meaningless: Files cannot be executed, and existing files cannot really be written to
- Permissions were not really enforced
  - ◆ Hadoop does not verify whether user's identity is genuine
  - ◆ Useful more to prevent accidental data corruption or casual misuse of information

# Where are things today?







- Since v.20.20x, Hadoop has some security
  - ◆ Kerberos RPC (SASL/GSSAPI)
  - ◆ HTTP SPNEGO authentication for web consoles
  - ◆ HDFS file permissions actually enforced
  - ◆ Various kinds of delegation tokens
  - ◆ Network encryption
  - ◆ For more details, see:  
<https://issues.apache.org/jira/secure/attachment/12428537/security-design.pdf>
- Big changes are coming
  - ◆ Project Rhino (e.g., encrypted data at rest)



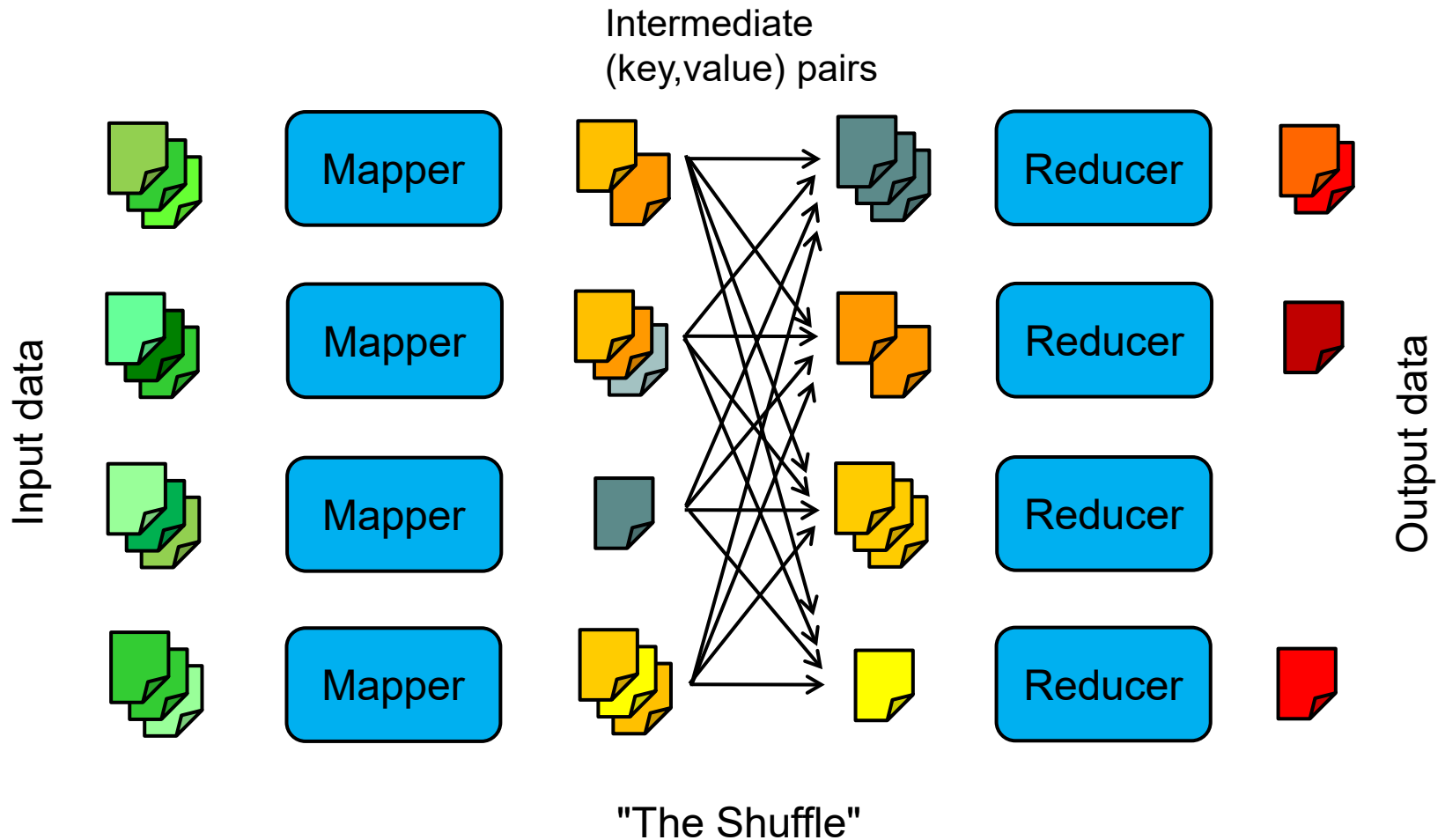
# Recap: HDFS

- **HDFS: A specialized distributed file system**
  - ◆ Good for large amounts of data, sequential reads
  - ◆ Bad for lots of small files, random access, non-append writes
- **Architecture: Blocks, namenode, datanodes**
  - ◆ File data is broken into large blocks (64MB default)
  - ◆ Blocks are stored & replicated by datanodes
  - ◆ Single namenode manages all the metadata
  - ◆ Secondary namenode: Housekeeping & (some) redundancy
- **Usage: Special command-line interface**
  - ◆ Example: `hadoop fs -ls /path/in/hdfs`
- **More details: Chapter 3 of your textbook**

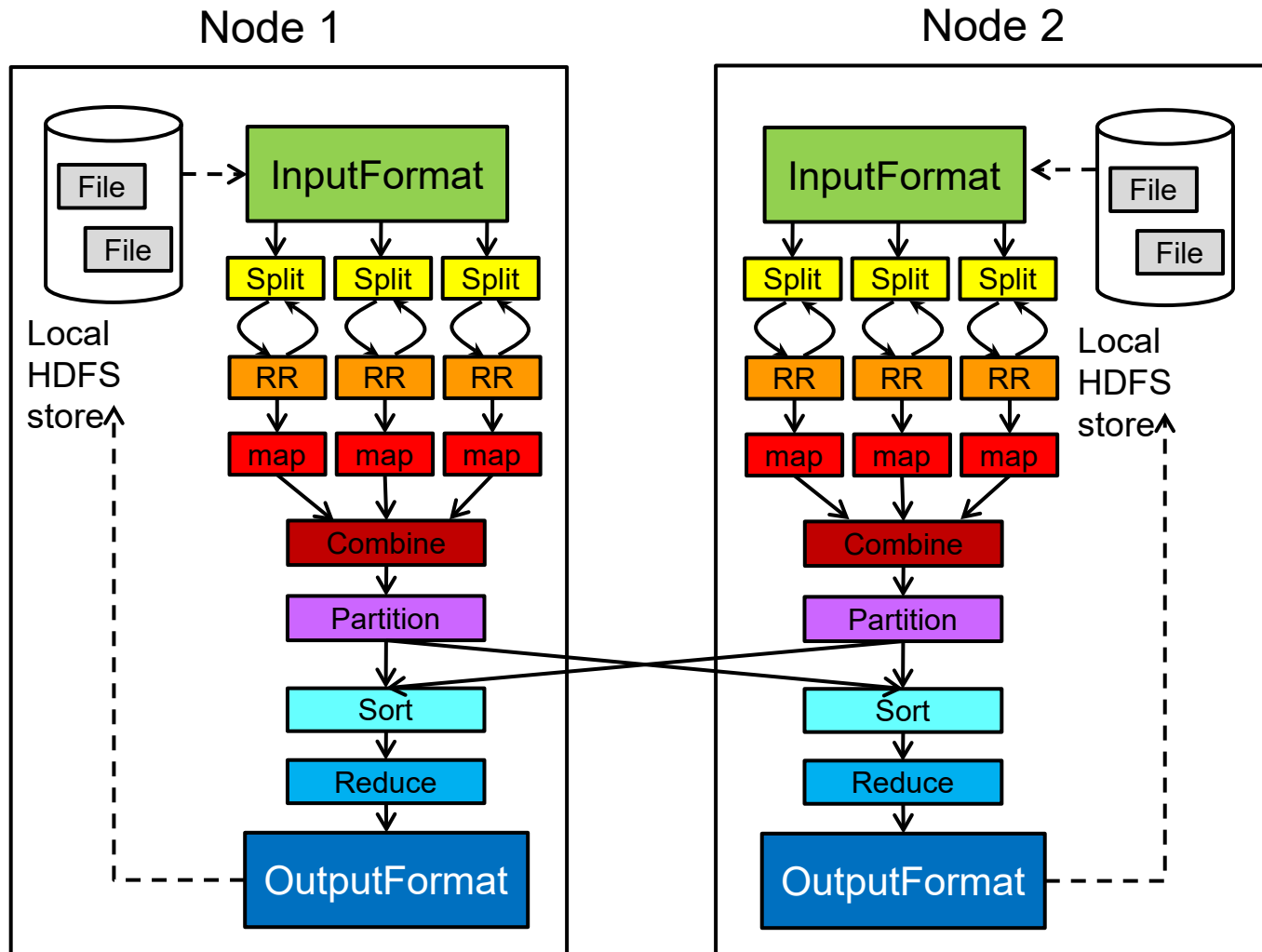
# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job 
- Hadoop Distributed File System (HDFS)
  - ◆ Node types; read and write operation 
  - ◆ Accessing data in HDFS 
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ... 
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution

# Recap: High-level dataflow

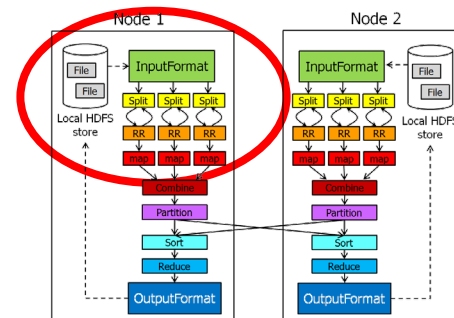


# Detailed dataflow in Hadoop



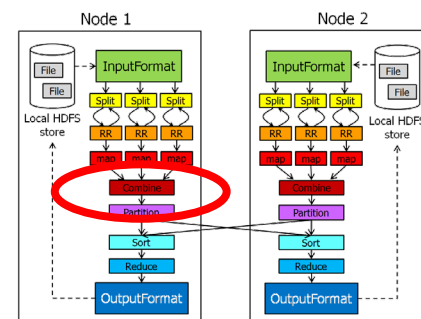
# Input Format

- Defines which input files should be read, and how
  - ◆ Defaults provided, e.g., TextInputFormat, DBInputFormat, KeyValueTextInputFormat...
- Defines InputSplits
  - ◆ InputSplits break file into separate tasks
  - ◆ Example: one task for each 64MB block (why?)
- Provides a factory for RecordReaders
  - ◆ RecordReaders actually read the file into (key,value) pairs
  - ◆ Default format, TextInputFormat, uses byte offset in file as the key, and line as the value
  - ◆ KeyValueInputFormat reads (key,value) pairs from the file directly; key is everything up to the first tab character



# Combiners

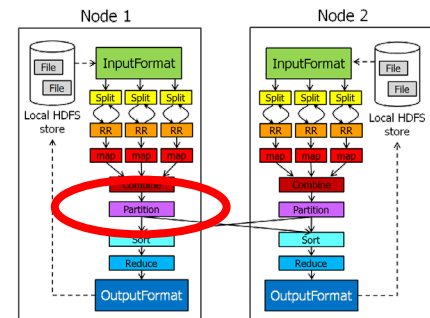
- Optional component that can be inserted after the mappers
  - ◆ Input: All data emitted by the mappers on a given node
  - ◆ Output passed to the partitioner



- Why is this useful?
  - ◆ Suppose your mapper counts words by emitting (xyz, 1) pairs for each word xyz it finds
  - ◆ If a word occurs many times, it is much more efficient to pass (xyz, k) to the reducer, than passing k copies of (xyz, 1)

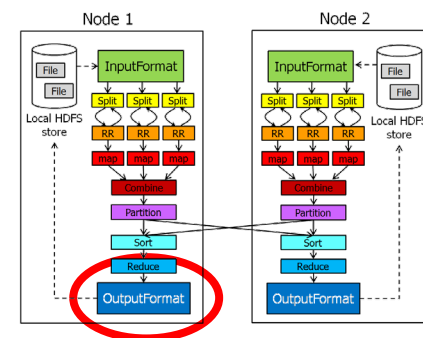
# Partitioner

- Controls which intermediate key-value pairs should go to which reducer
- Defines a partition on the set of KV pairs
  - ◆ Number of partitions is the same as the number of reducers
- Default partitioner (HashPartitioner) assigns partition based on a hash of the key



# Output Format

- Counterpart to InputFormat
- Controls where output is stored, and how
  - ◆ Provides a factory for RecordWriter
- Several implementations provided
  - ◆ TextOutputFormat (default)
  - ◆ DBOutputFormat
  - ◆ MultipleTextOutputFormat
  - ◆ ...











# Recap: Dataflow in Hadoop

- Hadoop has many components that are usually hidden from the developer
- Many of these can be customized:
  - ◆ InputFormat: Defines how input files are read
  - ◆ InputSplit: Defines how data portions are assigned to tasks
  - ◆ RecordReader: Reads actual KV pairs from input files
  - ◆ Combiner: Mini-reduce step on each node, for efficiency
  - ◆ Partitioner: Assigns intermediate KV pairs to reducers
  - ◆ Comparator: Controls how KV pairs are sorted after shuffle
- More details: Chapter 8 of your textbook

# Plan for today

- A brief history of Hadoop 
- Writing jobs for Hadoop
  - ◆ Mappers, reducers, drivers 
  - ◆ Compiling and running a job 
- Hadoop Distributed File System (HDFS)
  - ◆ Node types; read and write operation 
  - ◆ Accessing data in HDFS 
- Hadoop internals
  - ◆ Dataflow: Input format, partitioner, combiner, ... 
  - ◆ Fully distributed mode: Node types, setup
  - ◆ Fault tolerance; speculative execution



# Hadoop daemons (pre-YARN)

- TaskTracker

- ◆ Runs maps and reduces. One per node.

- JobTracker

- ◆ Accepts jobs; assigns tasks to TaskTrackers

- DataNode

- ◆ Stores HDFS blocks

A single node can run more than one of these!

- NameNode

- ◆ Stores HDFS metadata

- SecondaryNameNode

- ◆ Merges edits file with snapshot; "backup" for NameNode

# An example configuration



Small cluster



JobTracker



NameNode



Secondary  
NameNode



Medium cluster



JobTracker



NameNode



Secondary NameNode



TaskTracker



DataNode

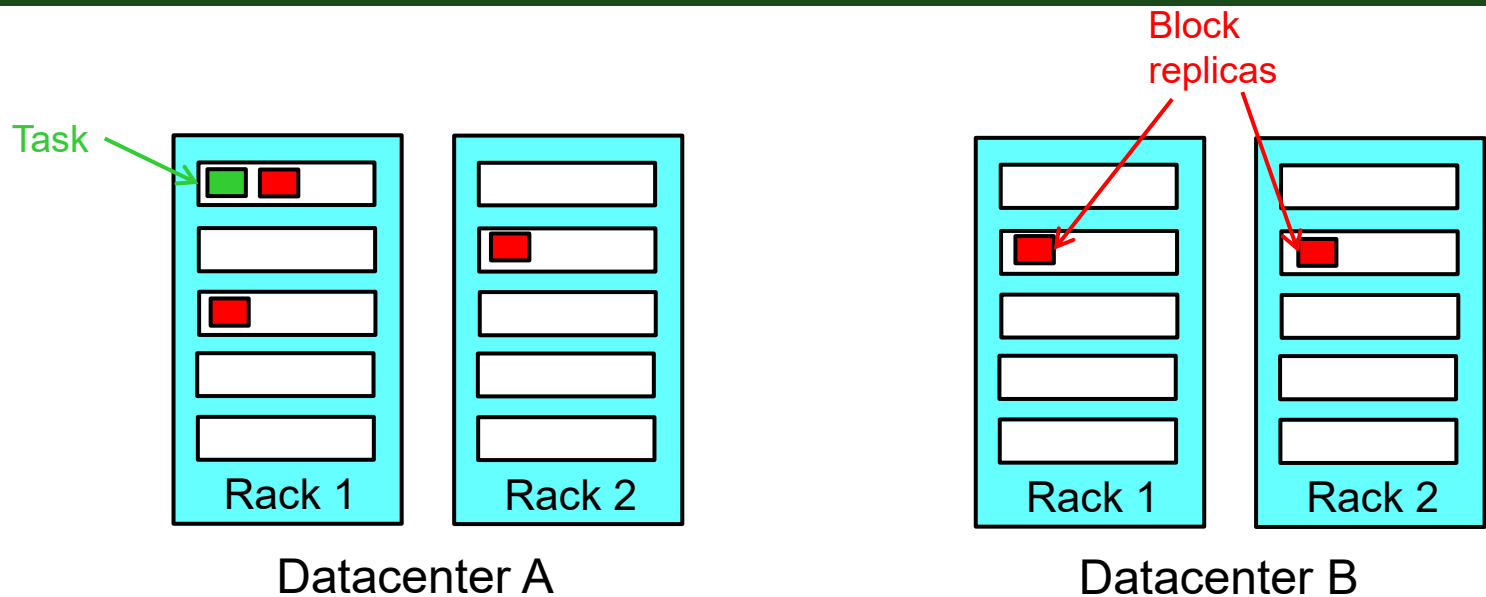
# Fault tolerance

- What if a node fails during a job?
  - ◆ JobTracker notices that the node's TaskTracker no longer responds; re-executes the failed node's tasks
- What specifically should be re-executed?
  - ◆ Depends on the phase the job was in
  - ◆ Mapping phase: Re-execute all maps assigned to failed node
  - ◆ Reduce phase: Re-execute all reduces assigned to the node
    - Is this sufficient?
    - No! Failed node may also have completed map tasks, and other nodes may not have finished copying out the results
    - Need to re-execute map tasks on the failed node as well!

# Speculative execution

- What if some tasks are much harder, or some nodes much slower, than the others?
  - ◆ Entire job is delayed!
- Solution: Speculative execution
  - ◆ If task is almost complete, schedule a few redundant tasks on nodes that have nothing else to do
  - ◆ Whichever one finishes first becomes the definitive copy; the others' results are discarded to prevent duplicates

# Placement and locality



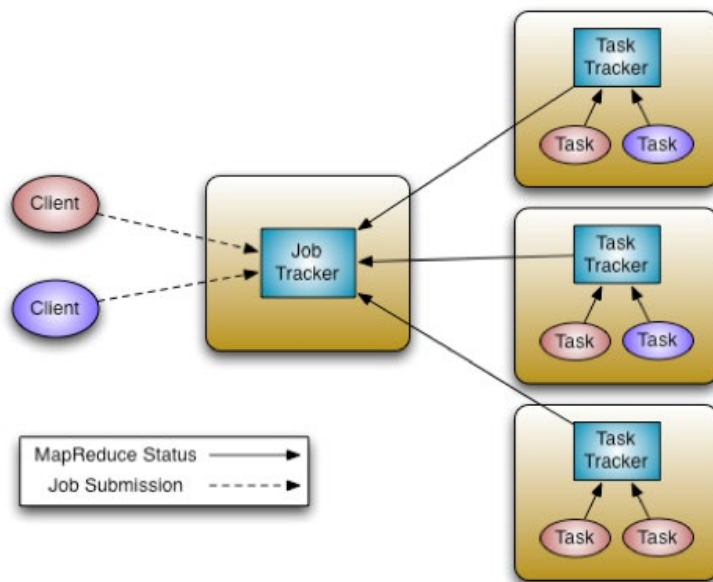
- Which of the replicated blocks should be read?
  - ◆ If possible, pick the closest one (reduces network load)
  - ◆ Distance metric takes into account: Nodes, racks, datacenters
- Where should the replicas be put?
  - ◆ Tradeoff between fault tolerance and locality/performance

# Changes in Hadoop 2.0

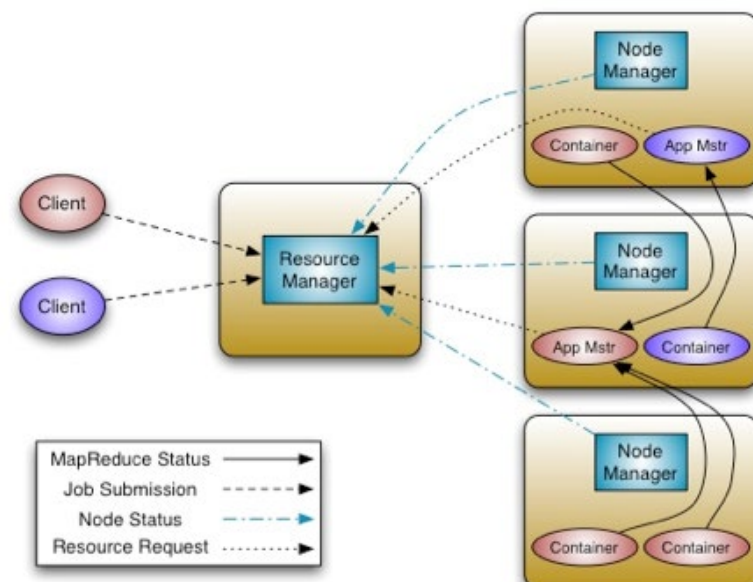
- In Hadoop 2.0/YARN, the two major responsibilities of the JobTracker have been split
  - ◆ Applications run in per-node **containers**
  - ◆ Global **ResourceManager**
    - Arbitrates resources among all the applications in the system based on their requirements (memory, CPU, ...)
    - A 'pure' scheduler. Scheduling algorithms are pluggable.
  - ◆ Per-application **ApplicationMaster**
    - Negotiates resources from the RM
    - Works with the NMs to execute/monitor the tasks
    - Responsible for status/progress tracking, fault tolerance, etc.
  - ◆ Per-node **NodeManager**
    - Launches the containers, monitors resource consumption
    - Reports to the ResourceManager
  - ◆ Backwards-compatible to existing MapReduce applications



# YARN architecture



Hadoop 1.0



Hadoop 2.0/YARN

## ■ Pros and cons of this architecture?

- ◆ Better scalability (one Application Master per application)
- ◆ Can support other frameworks (MPI, graph processing, ...)

# Recap: Distributed mode

## ■ Five important daemons:

- ◆ MapReduce daemons: JobTracker, TaskTracker
- ◆ HDFS daemons: DataNode, NameNode, Secondary NameN.
- ◆ Workers run TaskTracker+DataNode

## ■ Special features:

- ◆ Transparently re-executes jobs if nodes fail
- ◆ Speculatively executes jobs to limit impact of stragglers
- ◆ Rack-aware placement to keep traffic local



下节课:  
迭代式MR