软件架构与中间件

涂志莹 tzy_hit@hit.edu.cn

苏统华 thsu@hit.edu.cn

哈尔滨工业大学

软件架构与中间件

Software Architecture and Middleware

第3章

计算层的软件架构技术

高并发的处理

- PPC 和 TPC 模式
  - 它们的优点是实现简单
  - 缺点是都无法支撑高并发的场景，尤其是互联网发展到现在，各种海量用户业务的出现，PPC 和 TPC 完全无能为力。
- 应对高并发场景的单服务器高性能架构模式
  - Reactor： “来了一个事件我就有相应的反应”
  - Proactor： “来了事件我来处理，处理完了我通知你”

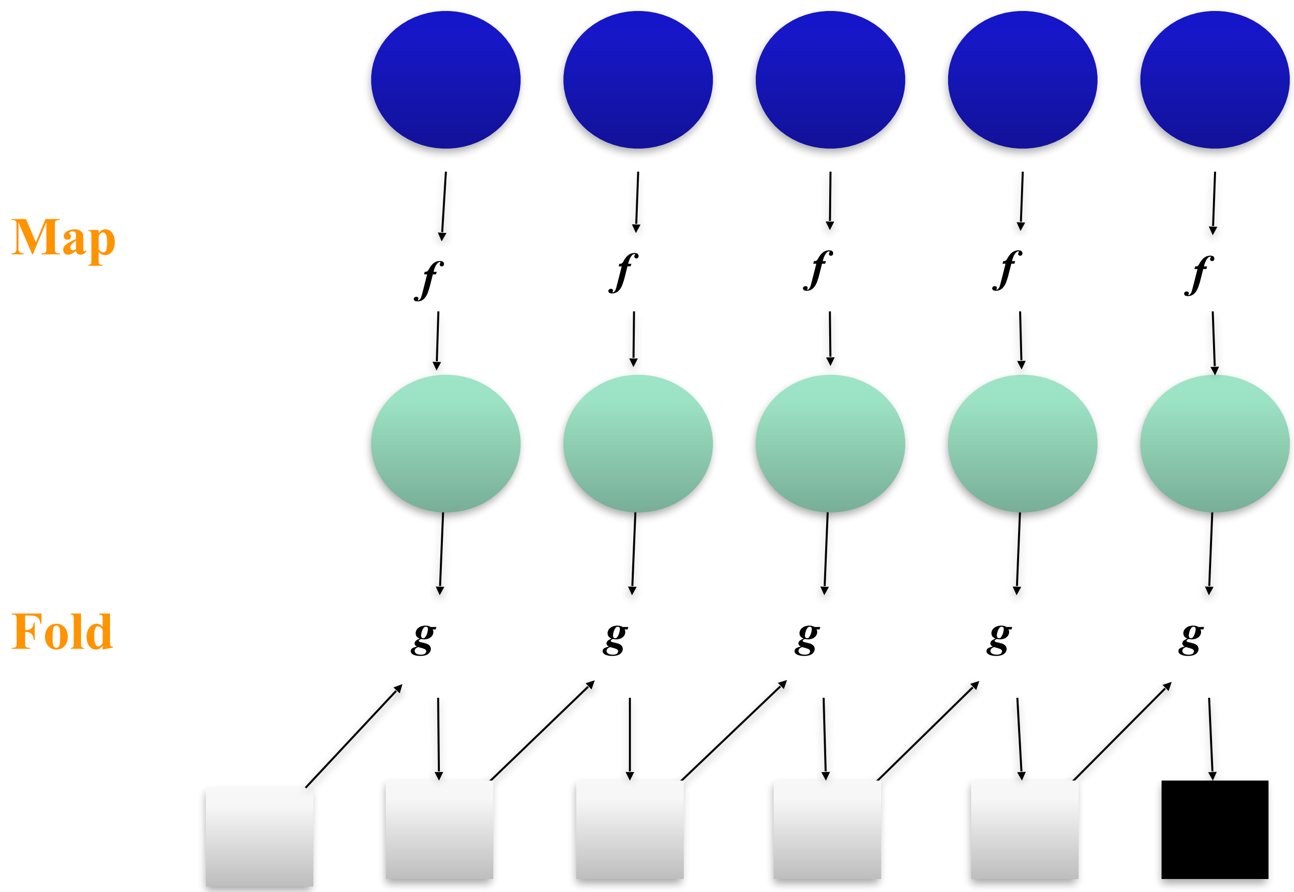
3.3.1 分布式编程模型

分布式编程模式

- 云计算上的编程模式
  - ➢必须十分简单
  - ➢必须保证后台复杂的并行执行和任务调度向用户和编程人员透明
- MapReduce
  - ➢Google提出的一种大规模数据处理的编程模型
  - ➢非分布式专业的编程人员也能够为大规模的集群编写应用程序
  - ➢应用程序编写人员只需要将精力放在应用程序本身，而关于集群的可靠性、可扩展性等问题则交由平台来处理

MapReduce

- Origin in Functional programming



MapReduce思想

- 例子
  - 问题
    - 数出一摞牌中有多少张黑桃

  - 方法1
    - 一张一张检查并且数出有多少张是黑桃
  - MapReduce方法
    - 给在座的所有同学中分配这摞牌
    - 让每个同学数自己手中的牌有几张是黑桃，然后把这个数目汇报给你
    - 你把所有同学告诉你的数字加起来，得到最后的结论

MapReduce

- MapReduce是一种针对超大规模数据集的编程模型和系统
- 用MapReduce开发出的程序可在大量商用计算机集群上并行执行、处理计算机的失效以及调度计算机间的通信
- MapReduce的基本思想
  - 用户写的两个程序：Map和Reduce
  - 一个在计算机集群上执行多个程序实例的框架

MapReduce

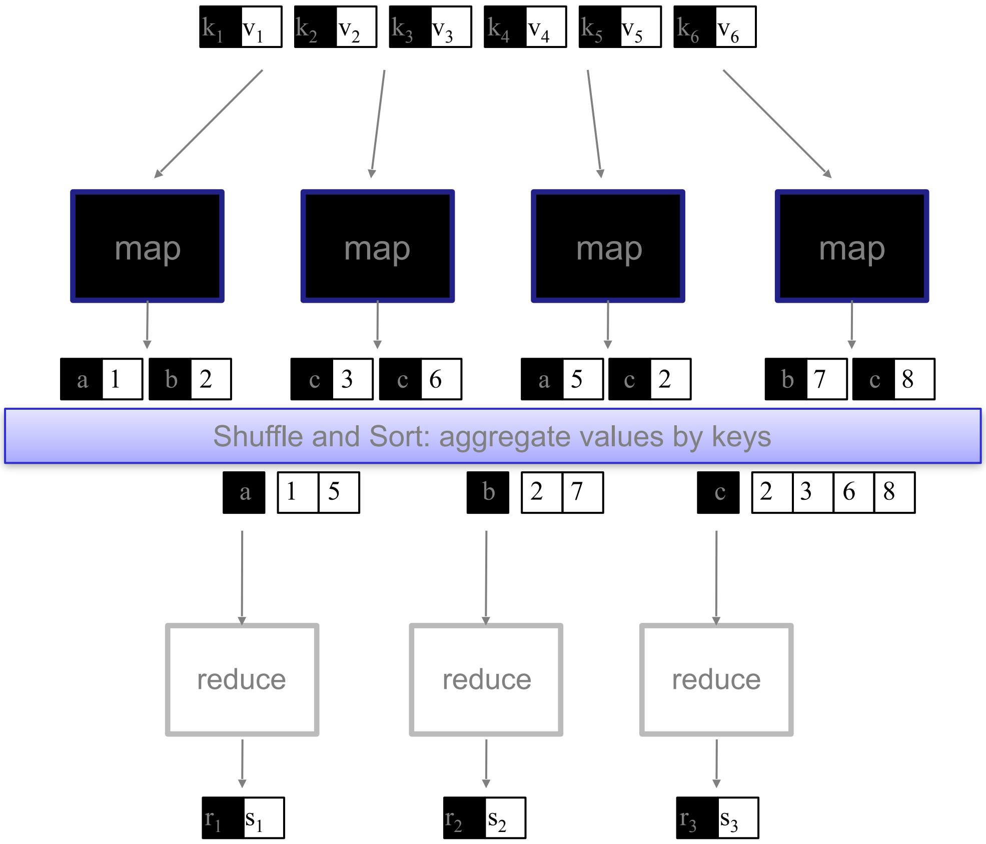
- Programmers specify two functions:

  map (k, v) → <k' , v' >*

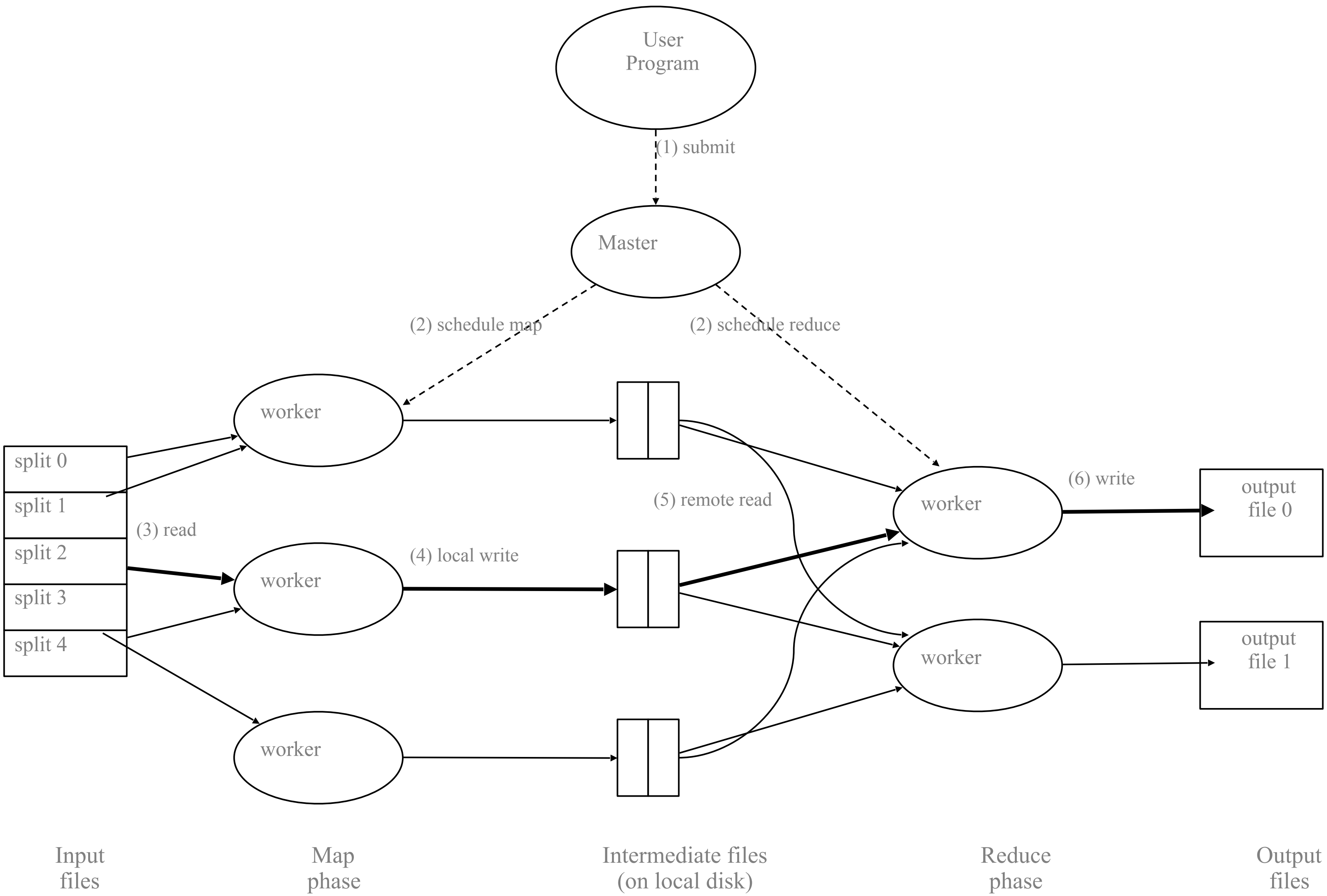
  reduce (k' , v') → <k' , v' >*

  - All values with the same key are sent to the same reducer
- The execution framework handles everything else...
  - Scheduling: Each MapReduce job is divided into smaller units called tasks. Then the tasks are assigned to different nodes.
  - Data/code co-location: MapReduce move the code to the node instead of moving the data to node unless data moving is unavoidable.
  - Synchronization: To "join up" the multiple concurrently running processes, we need synchronization such as to share intermediate results or otherwise exchange state information.
  - Error and fault handling: The MapReduce execution framework must accomplish all the tasks above in a environment where errors and faults are the norm, not the exception.(Bugs from both low-end commodity hardware & software.)

MapReduce


k_1 v_1 k_2 v_2 k_3 v_3 k_4 v_4 k_5 v_5 k_6 v_6
map
map
map
map
a 1 b 2
c 3 c 6
a 5 c 2
b 7 c 8
Shuffle and Sort: aggregate values by keys
a 1 5
b 2 7
c 2 3 6 8
reduce
reduce
reduce
r_1 s_1
r_2 s_2
r_3 s_3

MapReduce

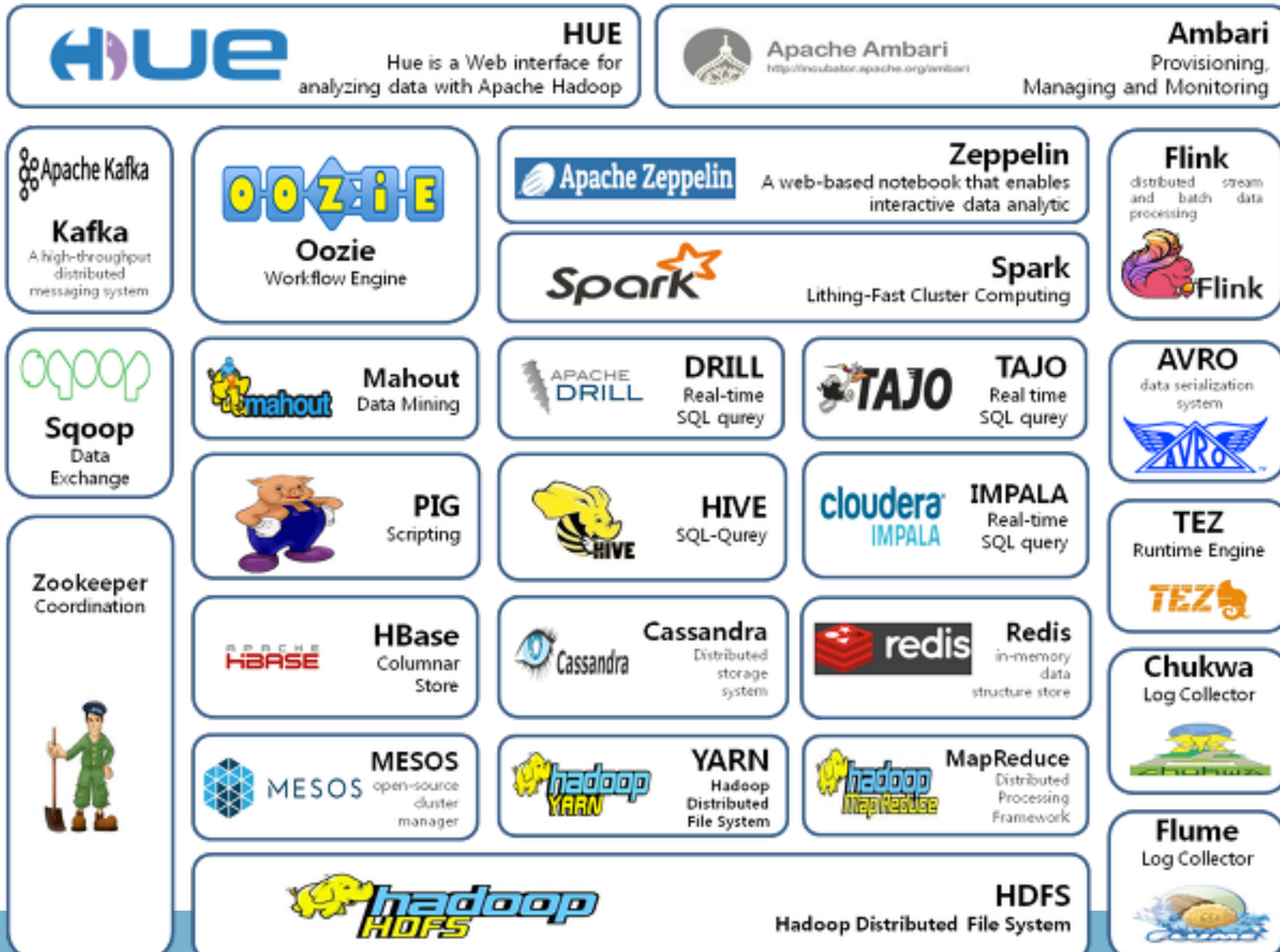

User Program
(1) submit
Master
(2) schedule map
(2) schedule reduce
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
worker
worker
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disk)
Reduce phase
Output files

MapReduce

- Google has a proprietary implementation in C++
  - Bindings in Java, Python
- Hadoop is an open-source implementation in Java
  - Original development led by Yahoo
  - Now an Apache open source project
  - Emerging as the de facto big data stack
  - Rapidly expanding software ecosystem
- Lots of custom research implementations
  - For GPUs, cell processors, etc.
  - Includes variations of the basic programming model

Most of these slides are focused on Hadoop

Hadoop Ecosystem



MapReduce: Recap

- Programmers must specify:

  map (k1, v1) → [(k2, v2)]

  reduce (k2, [v2]) → [(k3, v3)]

  - All values with the same key are reduced together
- Optionally, also:

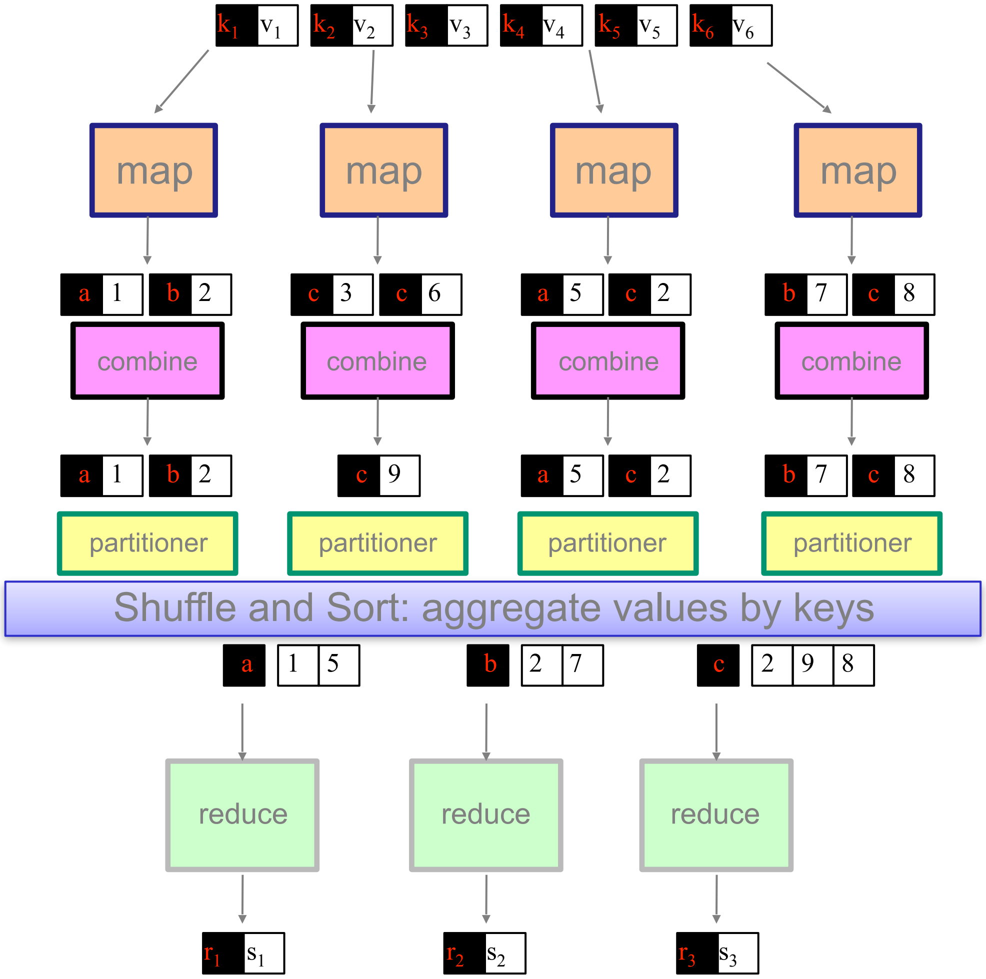
  partition (k2, number of partitions) → partition for k2

  - Often a simple hash of the key, e.g., hash(k’) mod n
  - Divides up key space for parallel reduce operations

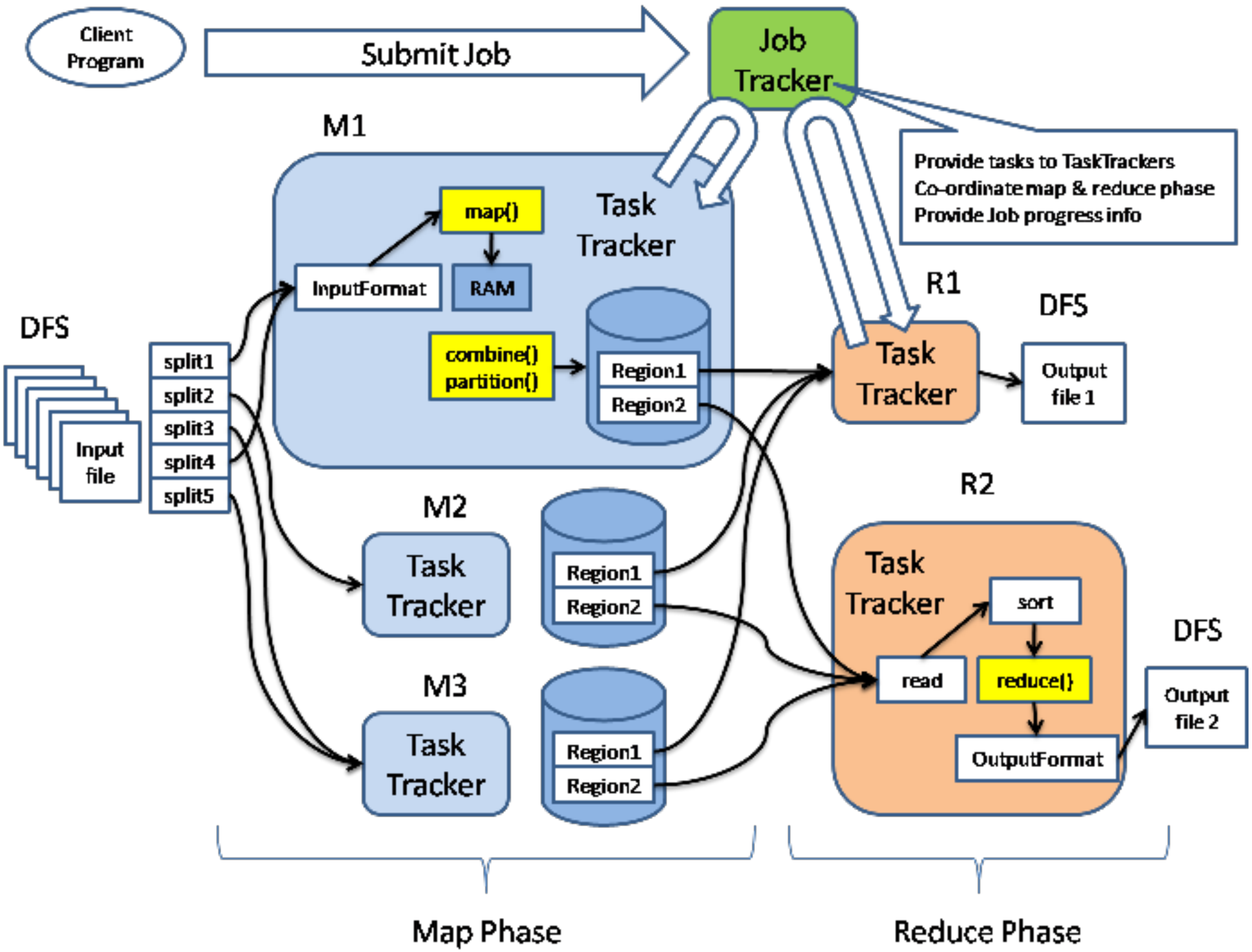
  combine (k2, [v2]) → [(k’ , v’)]

  - Mini-reducers that run in memory after the map phase
  - Used as an optimization to reduce network traffic
- The execution framework handles everything else…

MapReduce: Recap

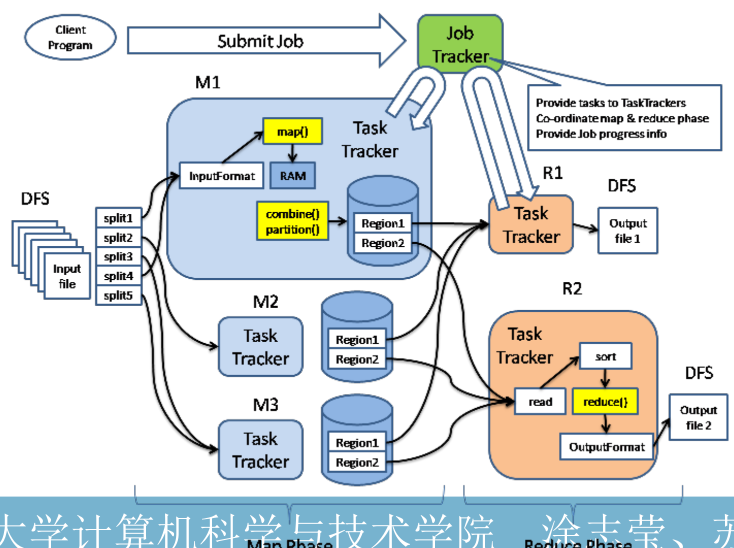


MapReduce in Hadoop



MapReduce in Hadoop

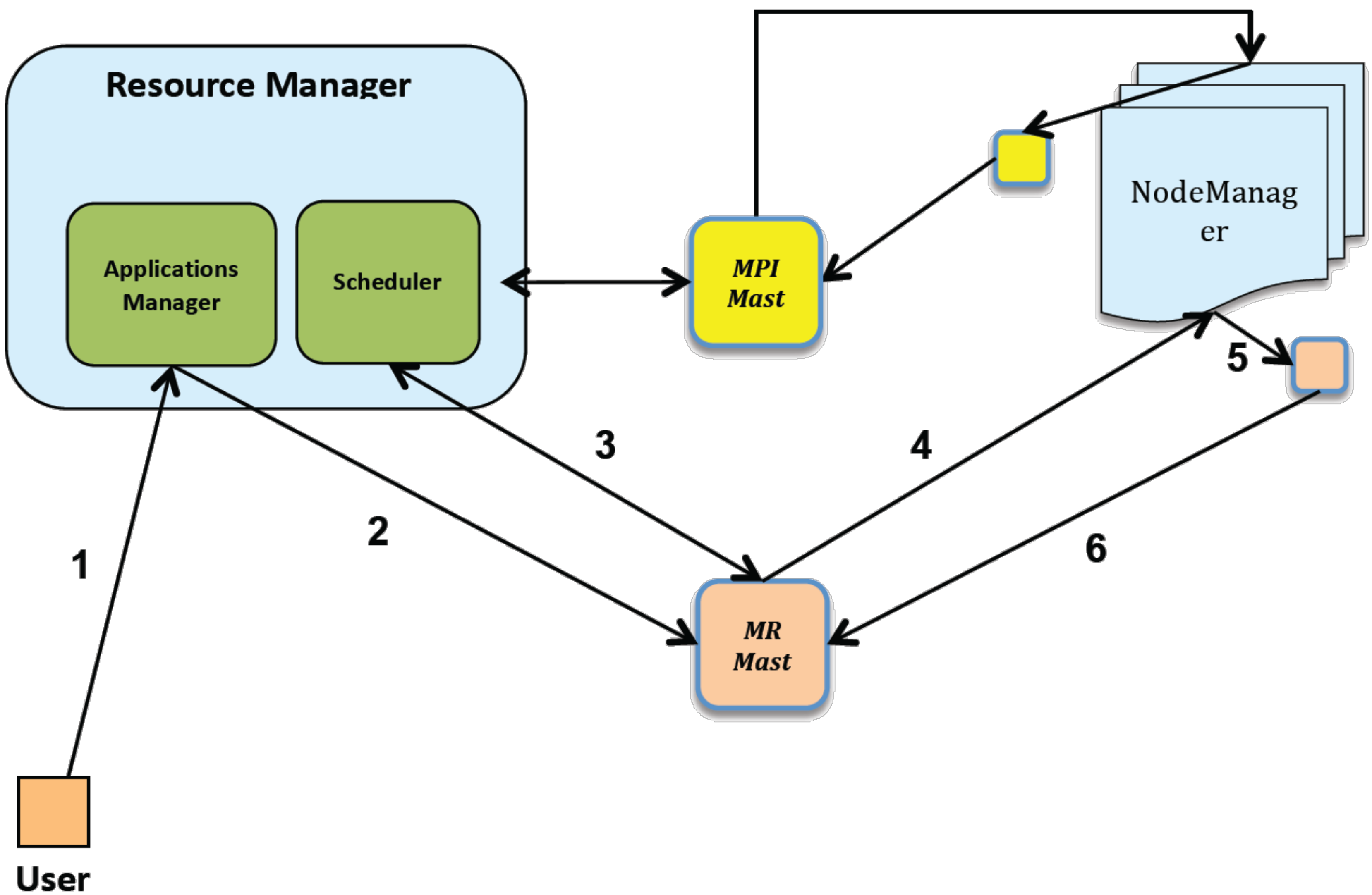
- **MapReduce 程序的流程及设计思路**
  - 首先提交一个 job，信息发给Job Tracker
  - Job Tracker 是框架的中心，定时与集群中机器通信，管理哪些程序跑在哪些机器上，管理所有 job 失败、重启等操作
  - TaskTracker 是 MapReduce 集群中每台机器都有的部分，它主要监视自己所在机器的资源情况，同时监视当前机器的 tasks 运行状况
  - TaskTracker 需要把这些信息通过 heartbeat 发送给 JobTracker
  - JobTracker 会搜集这些信息以给新提交的 job 分配运行在哪些机器上



MapReduce in Hadoop

- 主要的问题集中如下：
  - JobTracker 是 MapReduce 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 非常多时，会造成很大的内存开销：业界总结 Hadoop 的 MapReduce只能支持 4000 节点主机的上限
  - 在 TaskTracker 端，以 map/reduce task 的数目作为资源的表示过于简单，没有考虑到 cpu/ 内存的占用情况，如果两个大内存消耗的 task 被调度到了一块，很容易出现 溢出
  - 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
- MapReduce v2
  - 从 0.23.0 版本开始，Hadoop 的 MapReduce 框架完全重构，发生了根本的变化
  - 命名为YARN

MapReduce v2 in Hadoop: YARN

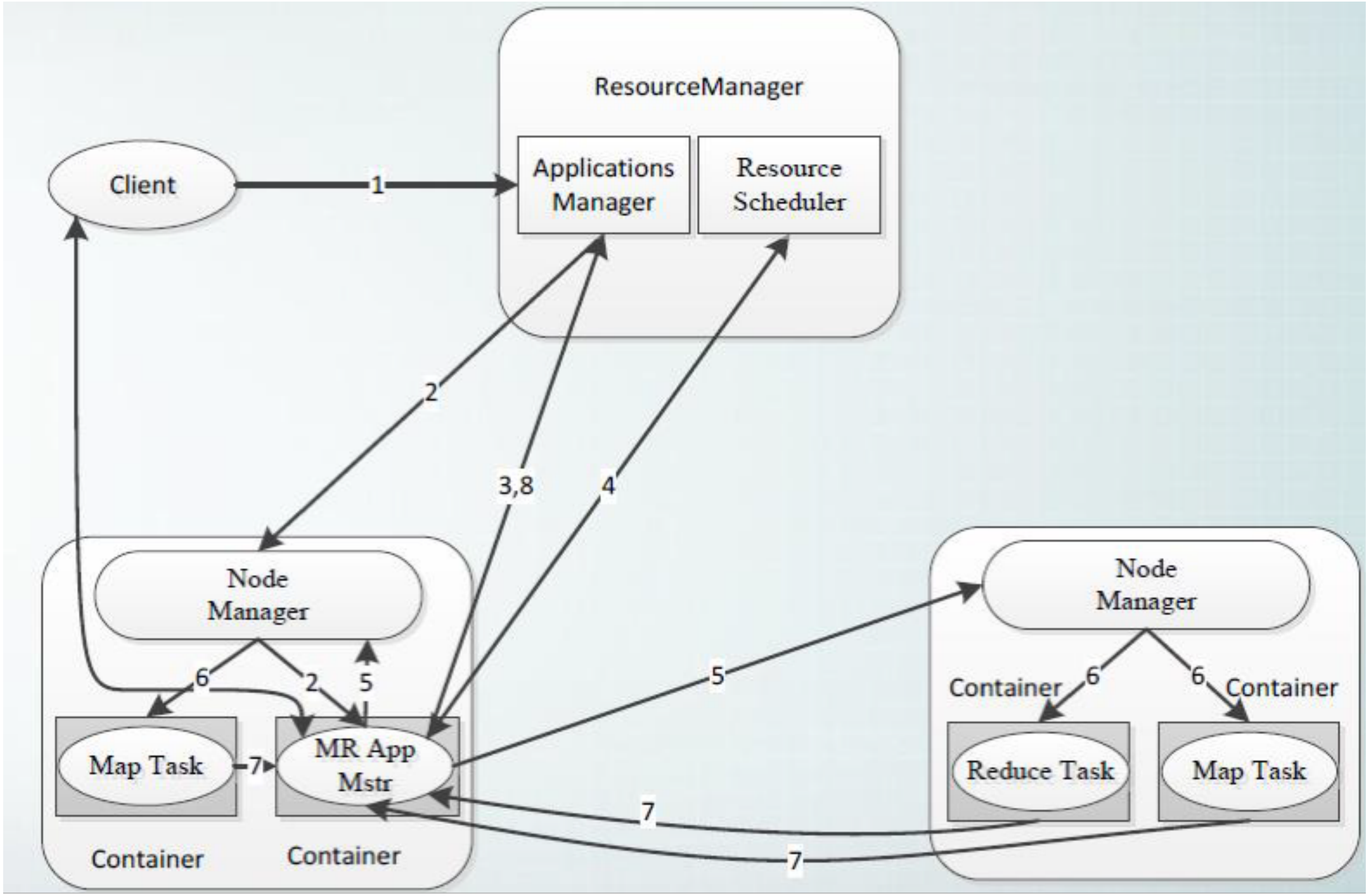


MapReduce v2 in Hadoop: YARN

- **设计优点**

1. 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、更优美
    - 另外，在新版中，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。
2. 能够支持不同的编程模型
3. 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
4. 既然资源表示成内存量，那就没有了之前的map slot/reduce slot分开造成集群资源闲置的尴尬情况了

MapReduce on YARN



示例1：Word Count

- Task
  - ➢ We have a huge text document
  - ➢ Count the number of times each distinct word appears in the file
  - ➢ e.g.

Big data is bigger
How to handle big data
Big data includes massive data

Big	2
How	1
big	1
bigger	1
data	4
handle	1
includes	1
is	1
massive	1
to	1

- Sample Application
  - ➢ Analyze web server logs to find popular URLs

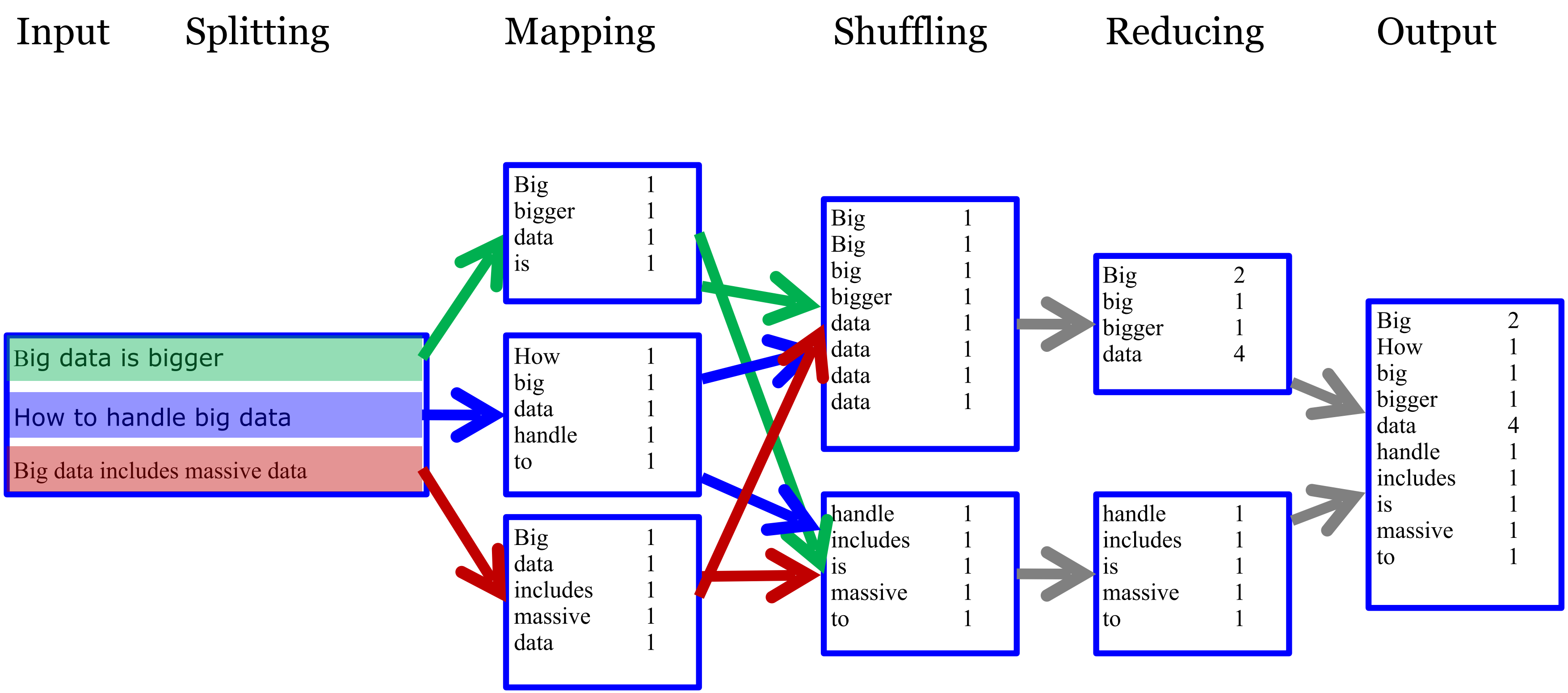
Word Count

- Pseudo Code in MapReduce

```
class Mapper
    method Map(docid a; doc d)
        for all term t ∈ doc d do
            EMIT(term t, count 1)
```

```
class Reducer
    method Reduce(term t; counts [c1, c2, …])
        sum:=0
        for all count c ∈ counts [c1, c2, …] do
            sum:=sum+c
    EMIT(term t, count sum)
```

Word Count


Input
Splitting
Mapping
Shuffling
Reducing
Output
Big data is bigger
How to handle big data
Big data includes massive data
Big 1
bigger 1
data 1
is 1
How 1
big 1
data 1
handle 1
to 1
Big 1
data 1
includes 1
massive 1
data 1
Big 1
Big 1
big 1
bigger 1
data 1
data 1
data 1
data 1
handle 1
includes 1
is 1
massive 1
to 1
Big 2
big 1
bigger 1
data 4
handle 1
includes 1
is 1
massive 1
to 1
Big 2
How 1
big 1
bigger 1
data 4
handle 1
includes 1
is 1
massive 1
to 1

示例2：Word Pairs(词对) Count

- Each mapper takes a sentence:
  - Generate all co-occurring term pairs
  - For all pairs, emit (a, b) → count
  - Use combiners!
- Reducer sums up counts associated with these pairs
- Advantages
  - Easy to implement, easy to understand
- Disadvantages
  - Lots of pairs to sort and shuffle around

Word Pair Count: Stripes

- Idea: group together pairs into an associative array (关联数组)

$$
\begin{aligned}
(a, b) &\rightarrow 1 \\
(a, c) &\rightarrow 2 \\
(a, d) &\rightarrow 5 \\
(a, e) &\rightarrow 3 \\
(a, f) &\rightarrow 2
\end{aligned}
\quad \Longrightarrow \quad a \rightarrow \{\, b{:}\ 1,\ c{:}\ 2,\ d{:}\ 5,\ e{:}\ 3,\ f{:}\ 2 \,\}
$$

- Each mapper takes a sentence:
  - ➢ Generate all co-occurring term pairs
  - ➢ For each term, emit a → { b: count_b, c: count_c, d: count_d … }
- Reducers perform element-wise sum of associative arrays

$$
\begin{array}{rl}
 & a \rightarrow \{\, b{:}\ 1, \qquad\quad d{:}\ 5,\ e{:}\ 3 \qquad\quad \} \\
+ & a \rightarrow \{\, b{:}\ 1,\ c{:}\ 2,\ d{:}\ 2, \qquad\quad f{:}\ 2 \,\} \\
\hline
 & a \rightarrow \{\, b{:}\ 2,\ c{:}\ 2,\ d{:}\ 7,\ e{:}\ 3,\ f{:}\ 2 \,\}
\end{array}
$$

Word Pair Count: Stripes

- Pseudo Code in MapReduce

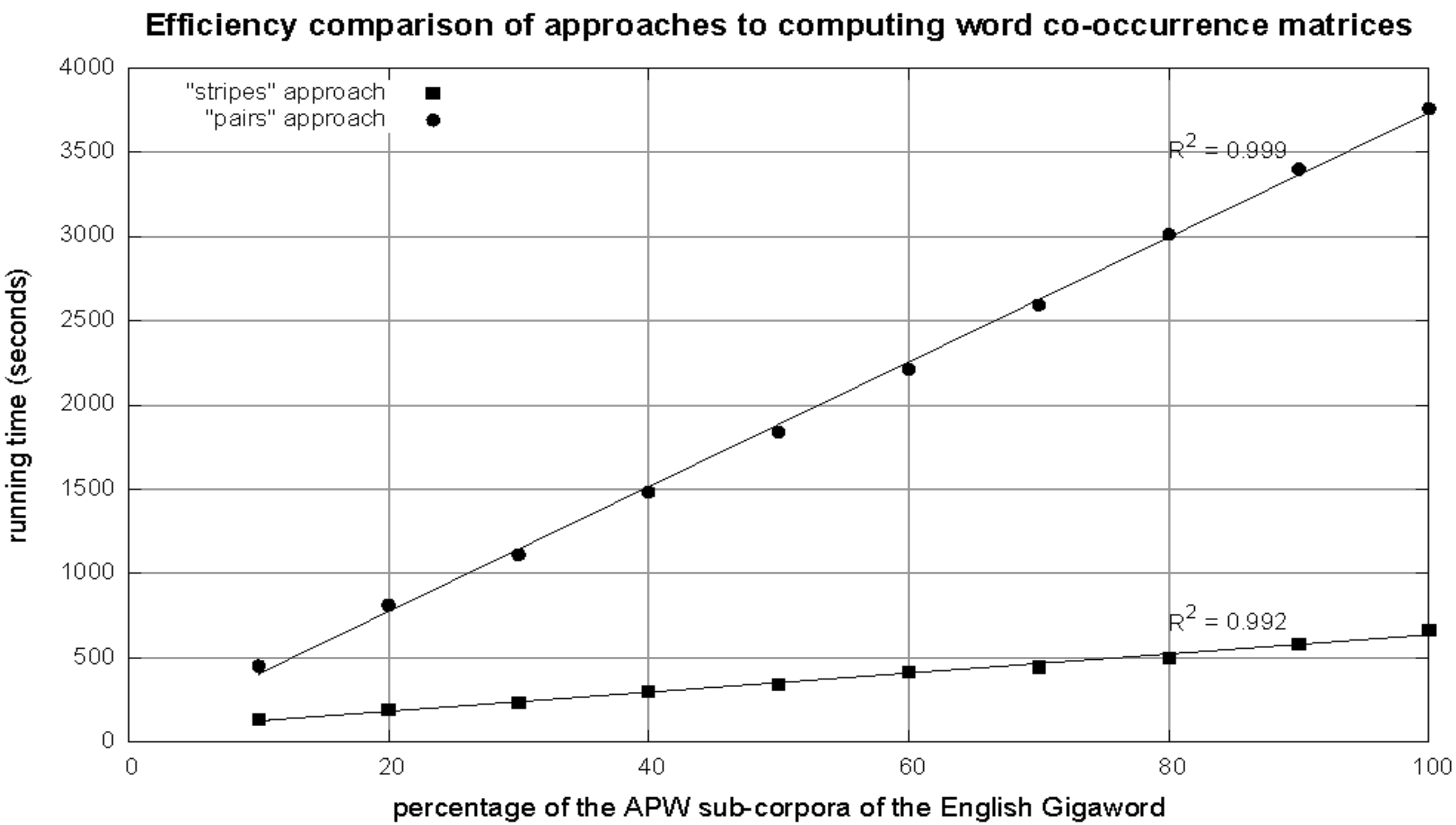
```
class Mapper
    method Map(docid a; doc d)
        for all term t ∈ doc d do
            H:=new AssociativeArray
            for all term u ∈ Neighbors(t) do
                H{u}:=H{u}+1
            EMIT(term t, Stripe H)
```

```
class Reducer
    method Reduce(term t; stripes[H_1, H_2, …])
        H_f:=new AssociativeArray
        for all stripe H ∈ stripes[H_1, H_2, …] do
            Sum(H_f,H)
        EMIT(term t, stripe H_f)
```

Word Pair Count: Stripes Analysis

- Advantages
  - Far less sorting and shuffling of key-value pairs
  - Make better use of combiners
- Disadvantages
  - More difficult to implement
  - Underlying object is more heavyweight
  - Fundamental limitation in terms of size of event space

Word Pair Count: Stripes Analysis



Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

第3章

计算层的软件架构技术

Thanks for Listening

涂志莹、苏统华
哈尔滨工业大学计算机学院
企业与服务计算研究中心