云计算

第6讲

MapReduce计算模型

任桐炜，李传艺

南京大学软件学院

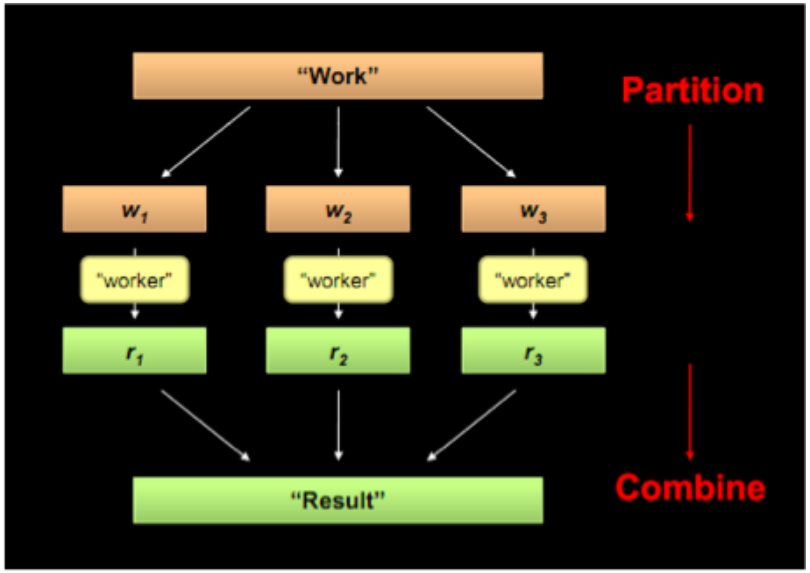
2017-10-16

设计要求

- 处理对象
  - 海量数据
  - 异构数据：结构化、半结构化、非结构化数据
- 应用需求
  - 有效管理
  - 高效分析

MapReduce模型

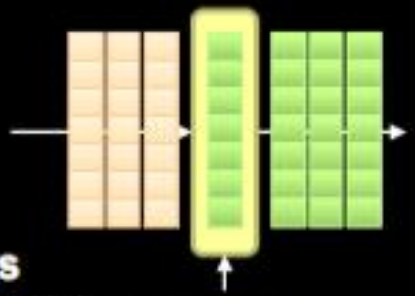
- 设计目标
  - 需要处理海量的数据（>1TB）
  - 在成百上千个CPU上并行处理
  - 较为容易地实现上述目标
- 基本思想
  - 分而治之



问题的复杂性

Fundamental issues

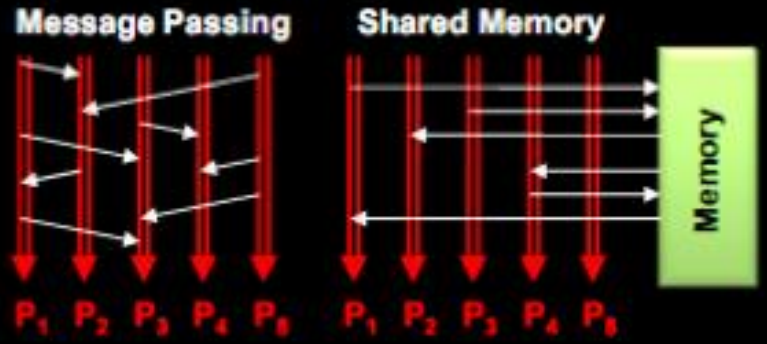
scheduling, data distribution, synchronization, inter-process communication, robustness, fault tolerance, …



Architectural issues

Flynn's taxonomy (SIMD, MIMD, etc.), network typology, bisection bandwidth
UMA vs. NUMA, cache coherence

Different programming models



Different programming constructs

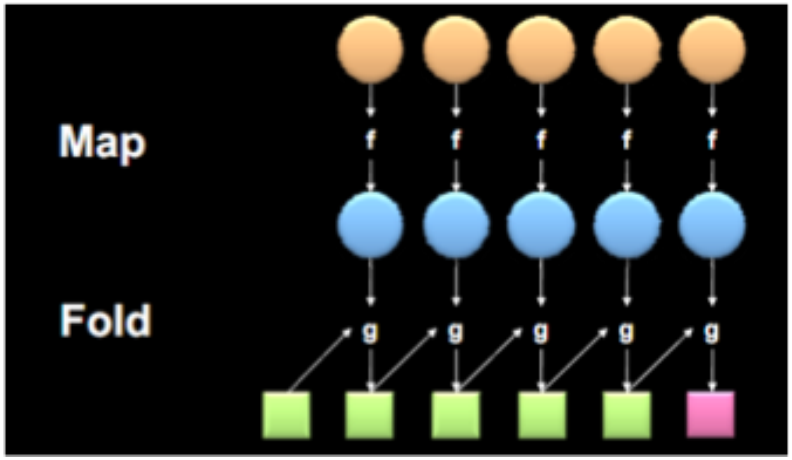
mutexes, conditional variables, barriers, …
masters/slaves, producers/consumers, work queues, …

Common problems

livelock, deadlock, data starvation, priority inversion…
dining philosophers, sleeping barbers, cigarette smokers, …

计算流程

- 遍历大规模的记录
- 从每个记录中抽取中间结果
- 对产生的中间结果重新排序
- 对排序后的中间结果集成
- 生成最终结果



Map map (in_key, in_value) -> (out_key, intermediate_value) list

Reduce reduce (out_key, intermediate_value list) -> out_value list

Word Count

```
map(String input_key, String input_value):
for each word w in input_value:
    EmitIntermediate(w, "1");
```

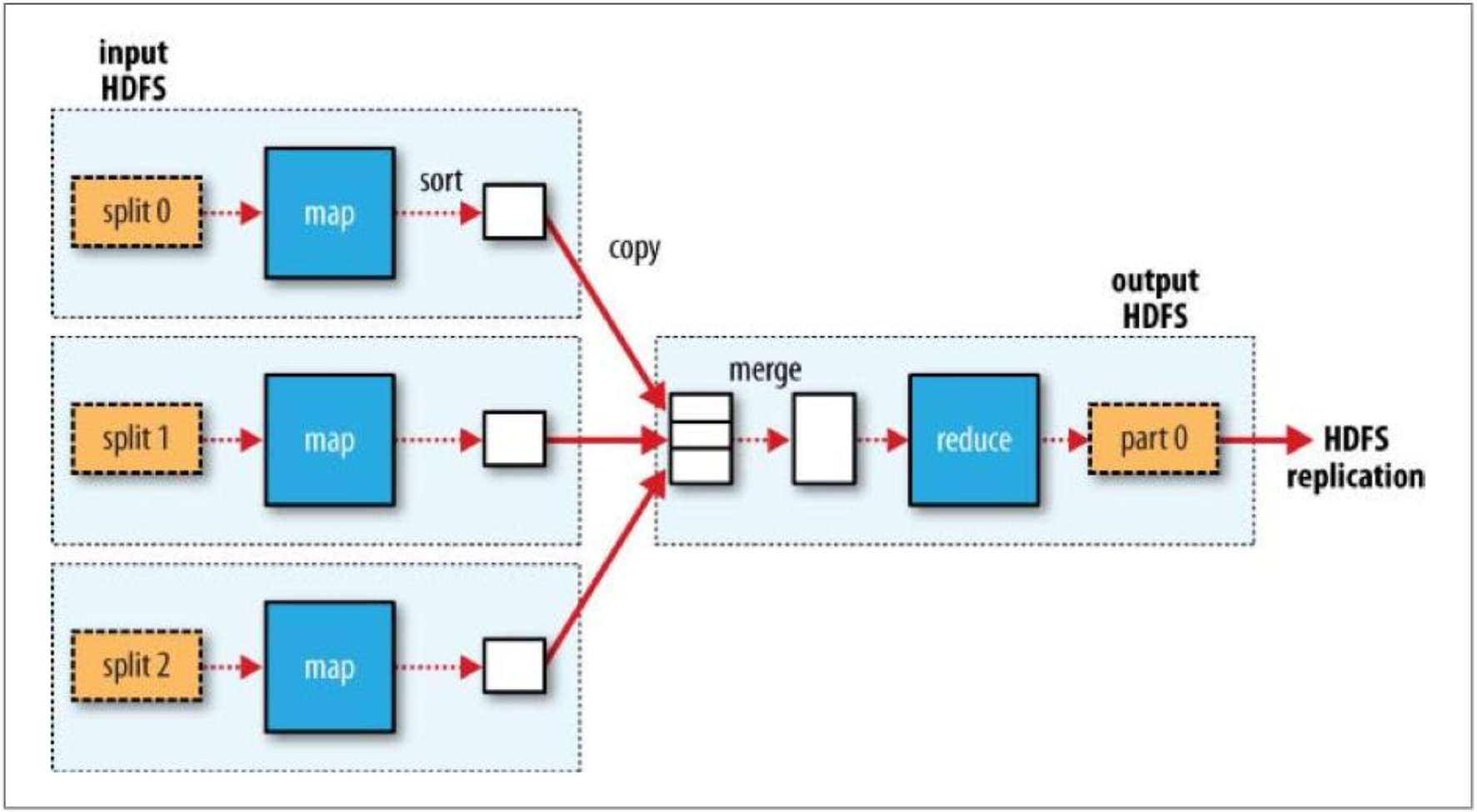
input_key: document name
input_value: document contents

```
reduce(String output_key, Iterator intermediate_values):
int result = 0;
for each v in intermediate_values:
    result += ParseInt(v);
Emit(AsString(result));
```

output_key: a word
output_values: a list of counts

- **WordCount 2.0:**
  http://hadoop.apache.org/common/docs/r0.18.2/cn/mapred_tutorial.html

简单的处理流程


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
output
HDFS
merge
reduce
part 0
HDFS
replication

数据分片

- 输入分片
  - **MapReduce**的输入数据会划分成等长的小数据块
- 分片数量较多：易于负载平衡
  - 处理每个分片所需要的时间少于处理整个输入数据所化的时间
- 分片大小过小
  - 管理分片的总时间和构建**map**任务的总时间将决定着作业的整个执行时间

本地计算

- Master根据数据所在的位置分配任务
  - 让Map和数据在同一个节点上
  - 至少处于同一个机架上
- Map处理数据的大小
  - 不大于数据块的大小（64MB）
  - 如果分片跨越2个数据块，则可能引起数据传输

中间结果存储

- **Map任务的结果写入本地硬盘，而非HDFS**
  - **Map输出的中间结果在程序运行结束后会删除**
  - **存放在HDFS中会浪费资源**

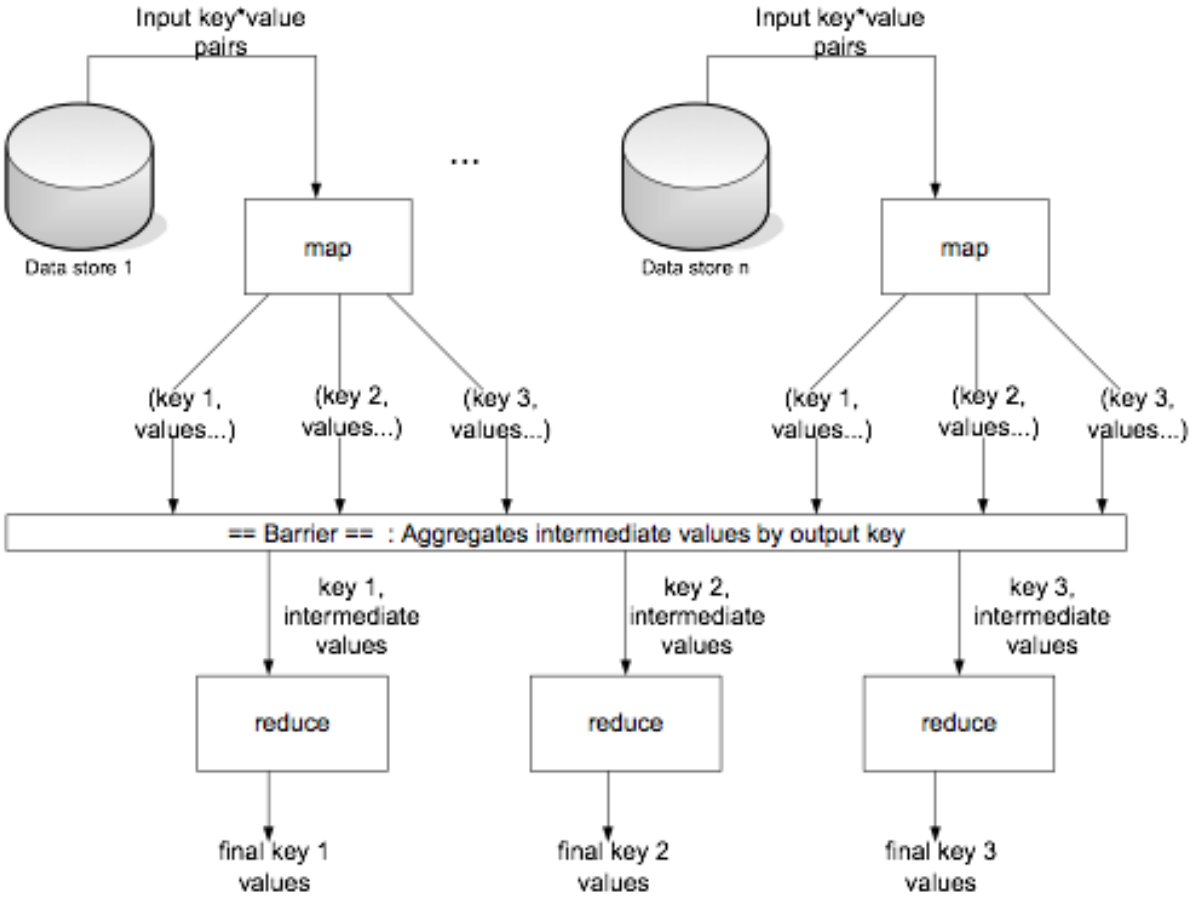
容错机制

- Master检测出错的Worker
  - 重新执行完成或进行中的map任务
  - 重新执行进行中的reduce任务
- Master检测到引起map出错的特定key
  - 在重新执行时忽略这些key

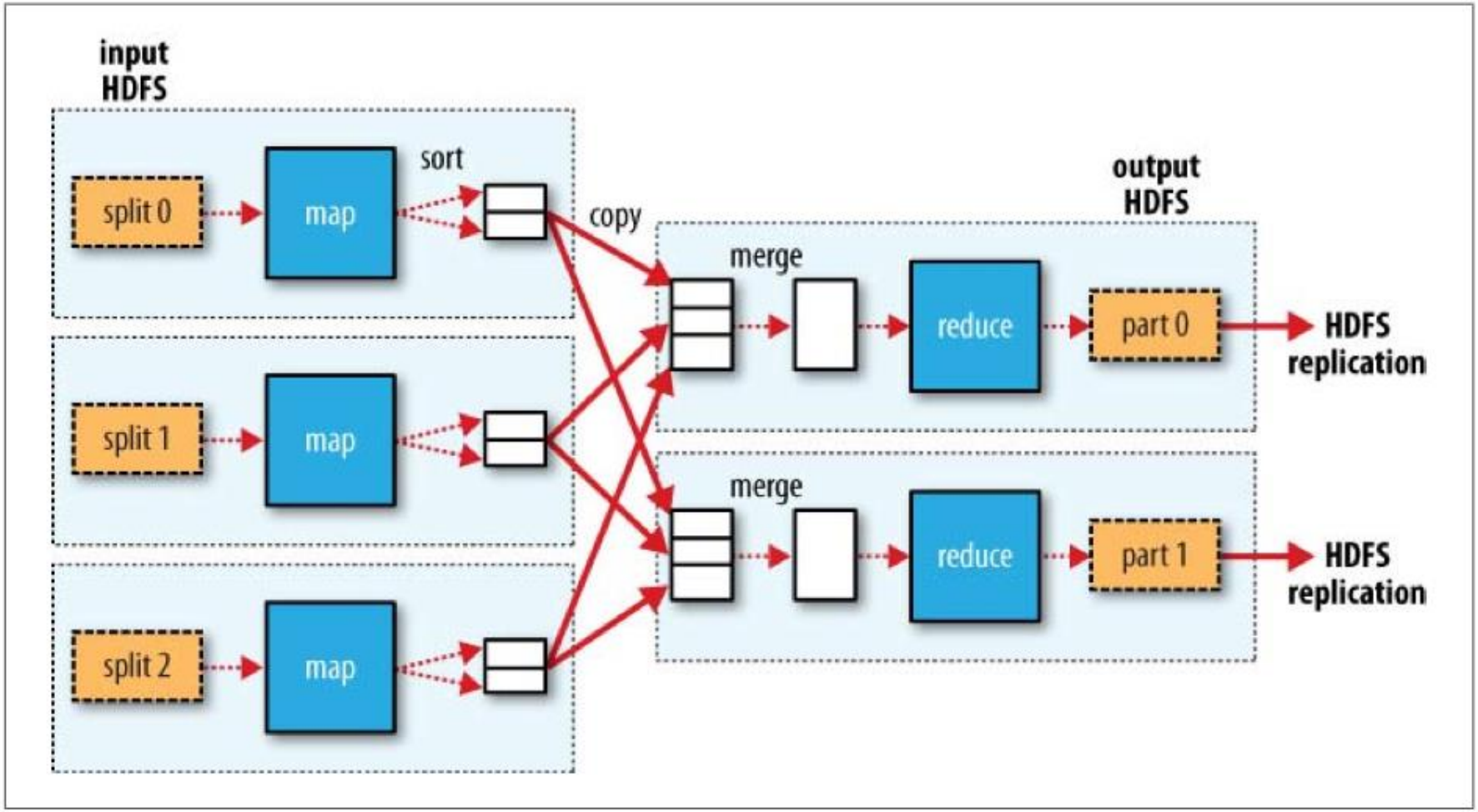
多个Reducer的处理流程

combine

partition



Partition


input
HDFS
split 0
map
sort
split 1
map
split 2
map
copy
merge
reduce
part 0
output
HDFS
HDFS
replication
merge
reduce
part 1
HDFS
replication

性能优化

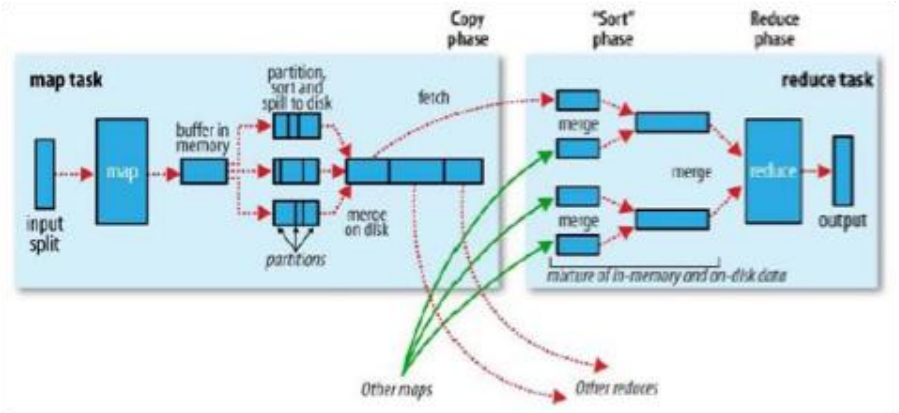
- 最慢的map任务会拖慢整个进程
  - Reduce任务需要等所有map任务结束后才能开始
  - Master重复分派较慢的map任务，并以最先完成的为结果
- Combine和map在同一个节点上执行
  - 将真正reduce阶段需要处理的数据量降至最低

MapReduce的优点

- 采用无共享大规模集群系统，具有良好的性价比和可伸缩性
  - 高性能计算/网格计算采用的是存储区域网络（SAN）组织的共享文件系统
- 模型简单，易于理解，易于使用，支持大多数大数据处理问题
- 只提供了一个过程性的编程接口，可以通过使用合适的查询优化和索引技术，提供相当好的数据处理性能

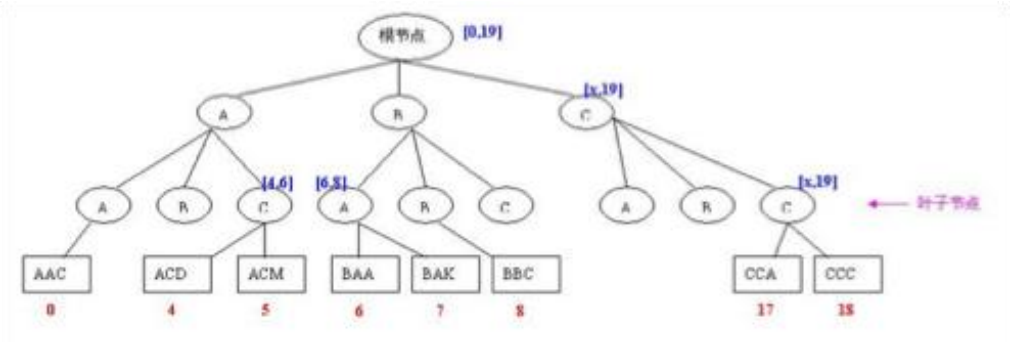
排序

- 问题
  - 按照key的顺序排
- 难点
  - 如何写Partitioner
- HashPartitioiner
  - 将mapper的输出通过hash函数映射到reducer
  - Reducer对输出的数据进一步排序



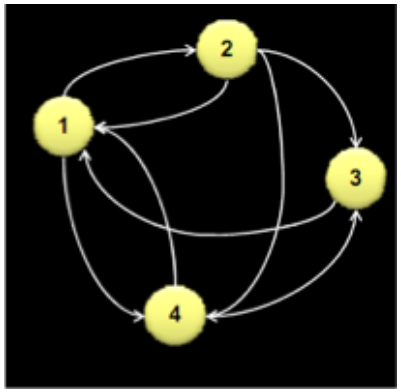
排序（续）

- 基于索引树的partition
  - 取样
    - 对每个split取样并排序，取出间隔平均的n-1个样
  - 构建索引树
    - 根据n-1个样，构建类似于B树的索引树
  - 分配
    - 前缀相同的key，被分配到同一个叶子节点
    - 每个叶子节点上可能有多个reducer



单源的最短路径

- 图：G = (V, E)
  - 两种表达：邻接矩阵，邻接列表
  - 问题：寻找从一个源点到一个或多个汇点的最短路径

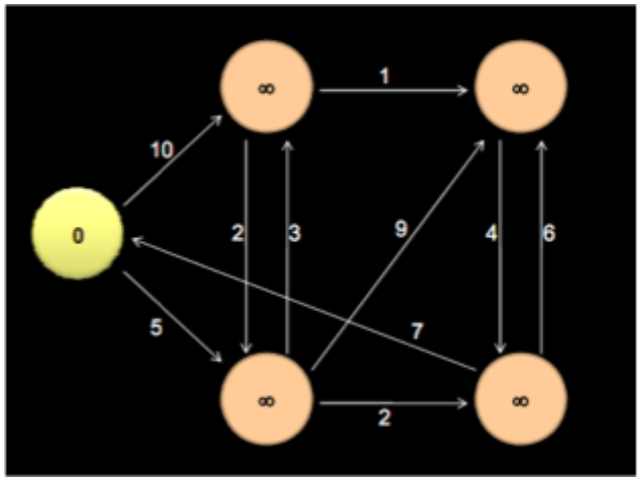


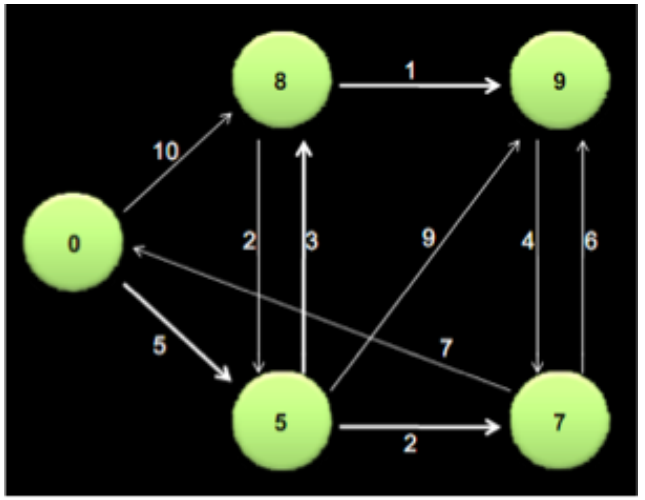
	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	0	1	0

1: 2, 4
2: 1, 3, 4
3: 2
4: 1, 3

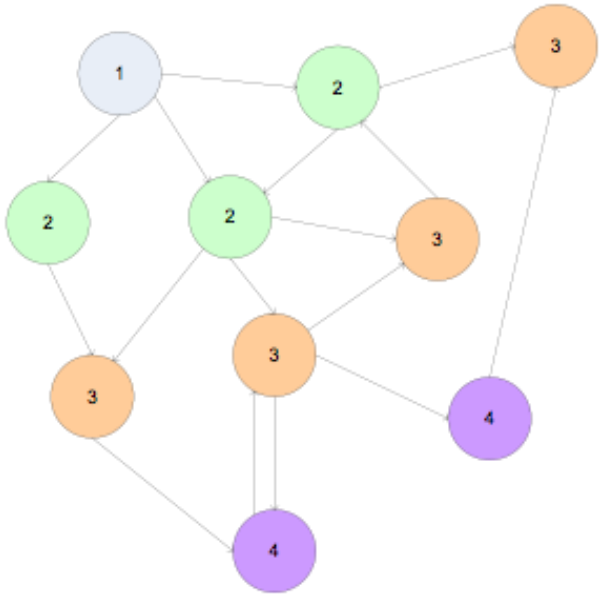
单源的最短路径（续）

- Dijkstra算法
  - 置集合S={2,3,...n}, 数组d(1)=0, d(i)=W1->i(1,i之间存在边) or + 无穷大(1,i之间不存在边)
  - 在S中，令d(j)=min{d(i),i ∈ S}，令S=S-{j}，若S为空集则算法结束，否则转3
  - 对全部i属于S,如果存在边j->i，那么置d(i)=min{d(i), d(j)+Wj->i}，转2





单源的最短路径（续）



- 广度优先搜索算法
  - 置集合S={1},源点的距离为：

    d(1)=0
  - 如果i与源点之间存在边，则距离为：

    d(i)=W1->i，S=S+{i}
  - 对于j，j的距离为： d(j)=min{d(i)+ Wi->j, i ∈ S}

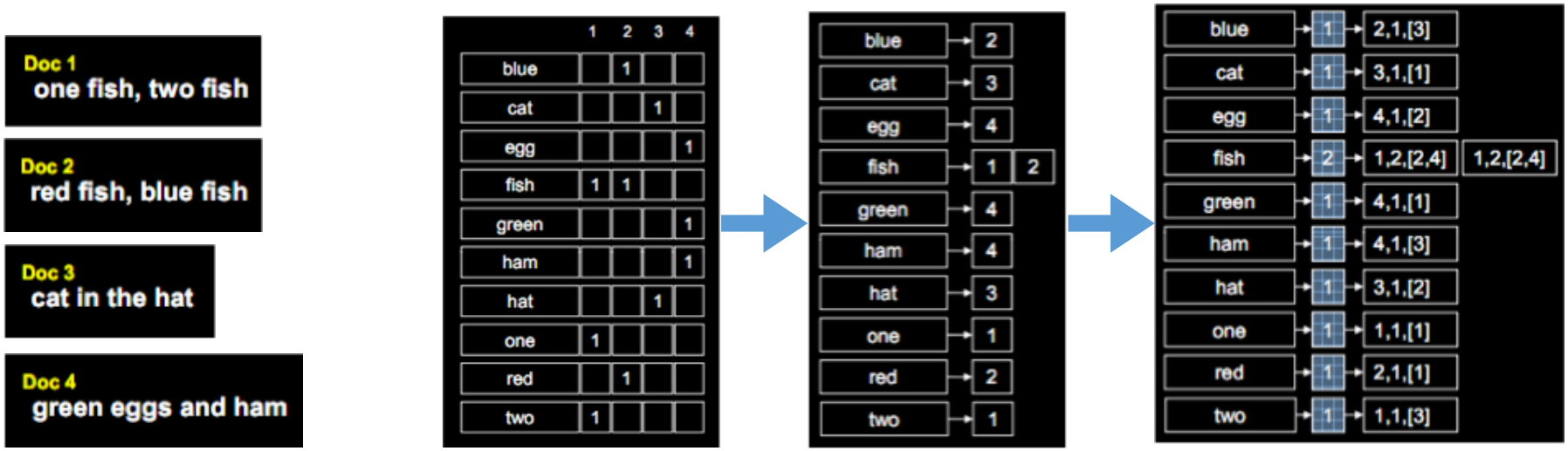
Mapper
- **Input:**
  key: node n
  value: D (distance from start), adjacency list
- **Output:** ∀p ∈ targets in adjacency list
  key = node p
  value = $D+W_{n->p}$

Reducer
- **Output:**
  key: node p
  value: minimum distance of p

网页的倒排索引

```
Procedure MAP(a,d)
    Initialize.AssociativeArray(H)
    for all t∈d do
        H{t}  H{t} + 1
    for all t∈H do
        Emit(t,<a,H{t}>)
```

```
Procedure
REDUCE(t,[<a1,f1>,<a2,f2>…])
Initialize.List(P)    for all <a,f> ∈
[<a1,f1>,<a2,f2>…] do
Append(P,<a,f>)    Sort(P)
Emit(t,P)
```



谢 谢