大数据算法

第八讲 MapReduce算法例析

哈尔滨工业大学
王宏志
wangzh@hit.edu.cn

本讲内容

8.1 连接(Join)算法

8.2 图算法

笛卡尔积运算

- 元组的串接（Concatenation）
  - 若$r = (r_1, \ldots, r_n)$，$s = (s_1, \ldots, s_m)$，则定义r与s的串接为：

$$\widehat{rs} = (r_1, \ldots, r_n, s_1, \ldots, s_m)$$

- 定义
  - 两个关系R，S，其度分别为n，m，则它们的笛卡尔积是所有这样的元组集合：元组的前n个分量是R中的一个元组，后m个分量是S中的一个元组
  - R×S的度为R与S的度之和， R×S的元组个数为R和S的元组个数的乘积

$$R \times S = \{ \widehat{rs} \mid r \in R \wedge s \in S \}$$

笛卡尔积运算

r

A	*B*
α	1
β	2

s

C	*D*	*E*
α	10	a
β	10	a
β	20	b
γ	10	b

r x *s*

A	*B*	*C*	*D*	*E*
α	1	α	10	a
α	1	β	19	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

笛卡尔积运算

- $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	19	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	20	a
β	2	β	20	b

θ连接

- 定义
  - 从两个关系的笛卡儿积中选取给定属性间满足一定条件的元组

$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{rs} \mid r \in R \wedge s \in S \wedge r[A]\theta s[B] \}$$

A,B为R和S上度数相等且可比的属性列

θ为算术比较符，为等号时称为等值连接

  - $R \underset{A\theta B}{\bowtie} S = \sigma_{r[A]\theta s[B]}(R \times S)$

示例

R

A	B	C
1	2	3
4	5	6
7	8	9

S

D	E
3	1
6	2

$R \bowtie S$
$B < D$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

自然连接

- 定义
  - 从两个关系的广义笛卡儿积中选取在相同属性列B上取值相等的元组。

$$R \bowtie S = \{ \widehat{rs}[\bar{B}] \mid r \in R \wedge s \in S \wedge r[B]=s[B] \}$$

  - 自然连接与等值连接的不同
    - 自然连接要在结果中去掉重复的属性，而等值连接则不必。

自然连接

r

A	*B*	*C*	*D*
α	**1**	α	**a**
β	2	γ	a
γ	4	β	b
α	**1**	γ	**a**
δ	2	β	b

s

B	*D*	*E*
1	**a**	α
3	a	β
1	**a**	γ
2	b	δ
3	b	$\in$

$r \bowtie s$

A	*B*	*C*	*D*	*E*
α	**1**	α	**a**	α
α	**1**	α	**a**	γ
α	**1**	γ	**a**	α
α	**1**	γ	**a**	γ
δ	2	β	b	δ

基于MapReduce的连接算法

- Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters *sigmod07*
- Semi-join Computation on Distributed File Systems Using Map-Reduce-Merge Model *Sac10*
- Optimizing joins in a map-reduce environment *VLDB09,EDBT2010*
- A Comparison of Join Algorithms for Log Processing in MapReduce *sigmod10*

Map-Reduce-Merge关系连接算法的实现

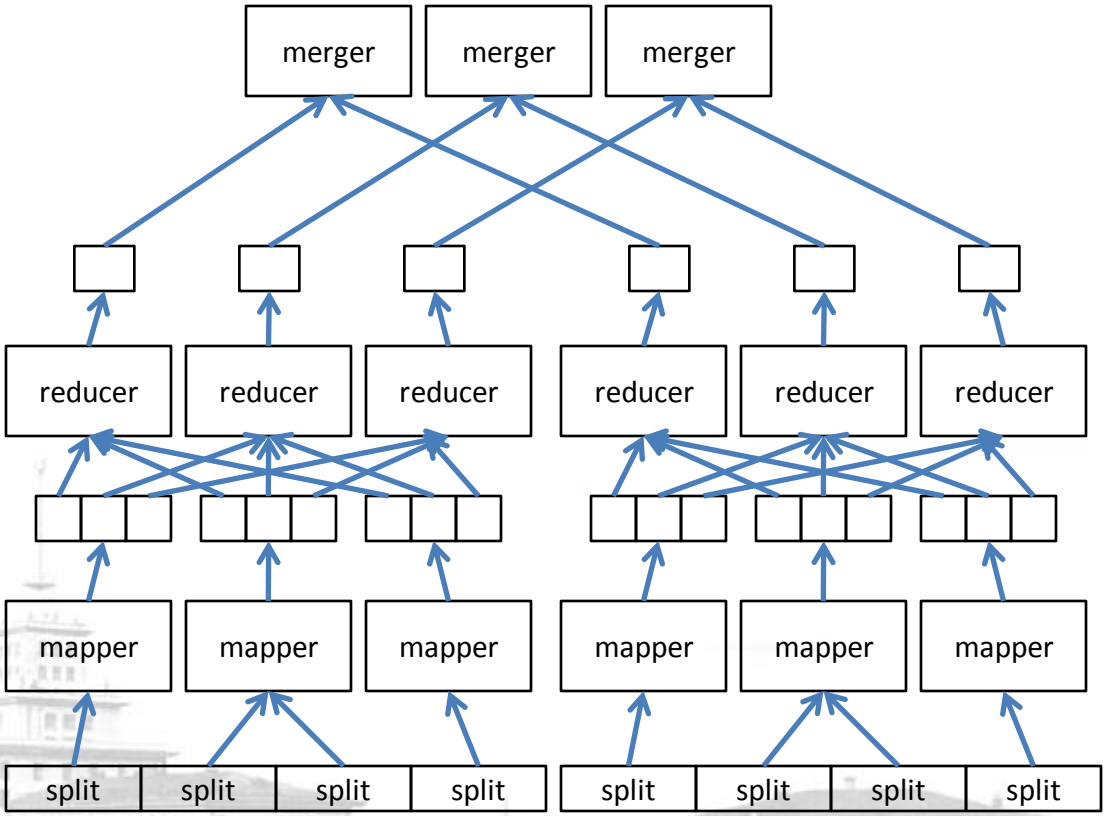
Sort-merger join	Map	区间partitioner , 生成排序的桶, 每个桶对应一个reducer
	Reduce	从所有mapper读取桶并且将其归并到一个排序的集合中
	Merge	从连个数据集合中读取排序的桶并执行sort-merge join
Hash join	Map	Hash partitioner, 桶哈希,每个桶对应一个reducer
	Reduce	从所有mapper中读取桶，使用hash表分组和聚集这些记录(使用和mapper相同的hash函数)，无需排序
	Merge	内存hash join
Block Nested loop join	Map	同hash join
	Reduce	同hash join
	Merge	Nested loop join

例子: Hash Join

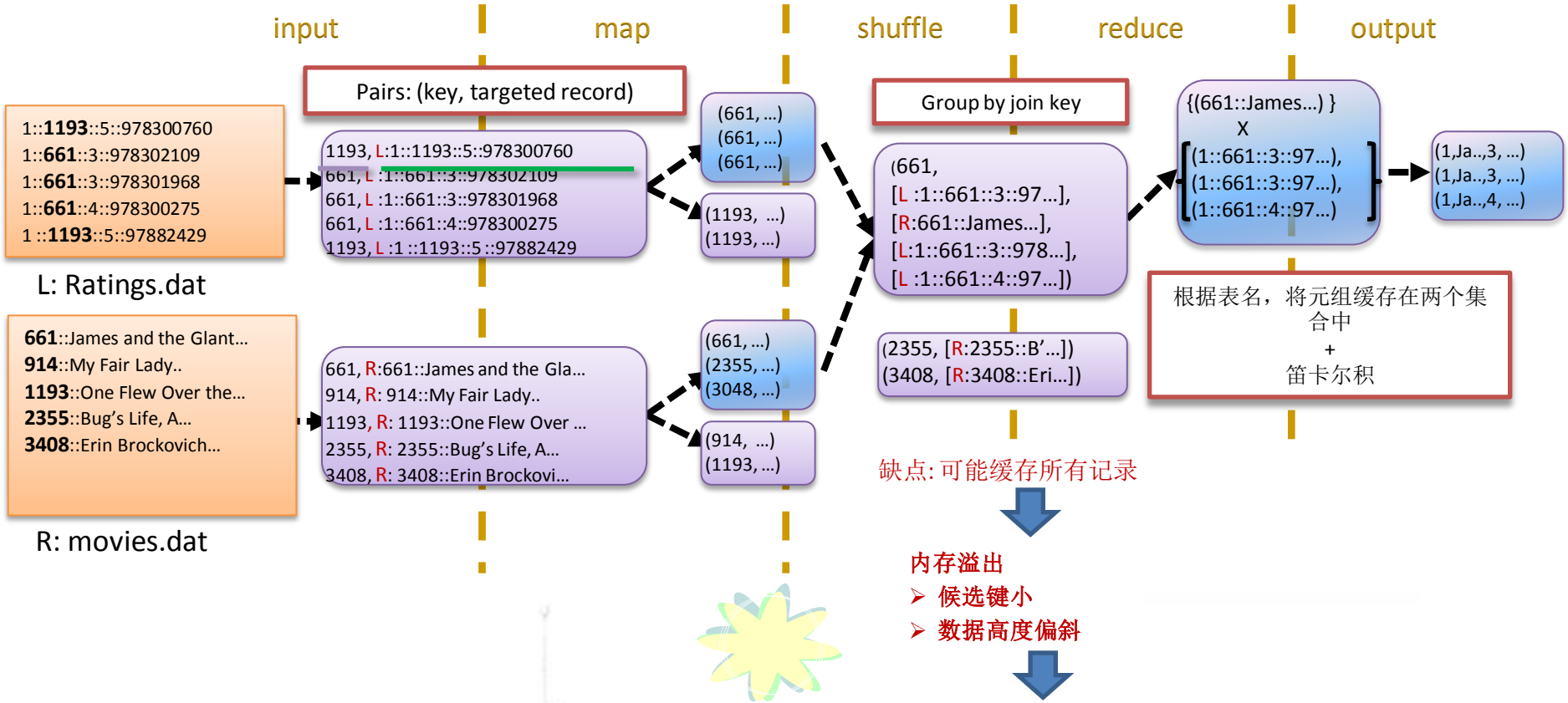
•从两个共享相同hash桶的reducer输出集合中读取数据s
•其中之一用于构建集合，另一个用于探测

从每个mapper中读取制定的块

使用哈希partitioner



repartition join(Hive)



函数	改进
Map函数	输出键变化为连接键和表名的组合
Partitioning函数	Hashcode仅从组合键的连接键计算
Grouping函数	仅根据连接键分组记录

MapReduce上的2路 Join

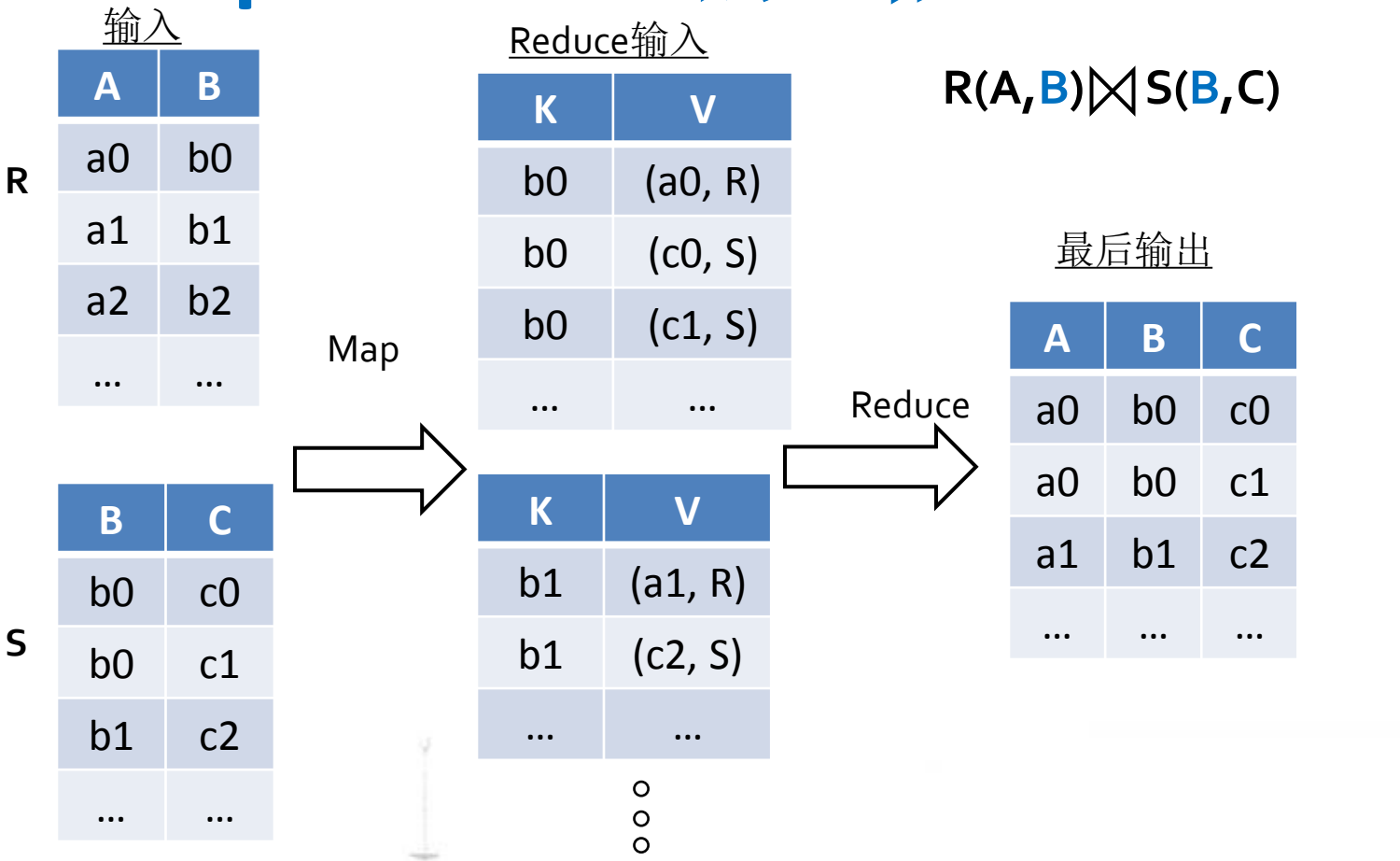
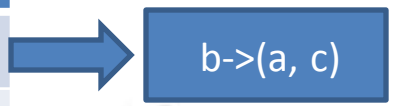


表	元组	map	Partition& sort
R	(a ,b)	b ->(a, R)	Hash(b) ->(a, R)
S	(b , c)	b ->(c, S)	Hash(b) ->(c, S)



一次处理多个连接

R(A,B)⋈ S(B,C)⋈ T(C,D)

输入

R

A	B
a0	b0
a1	b1
...	...

S

B	C
b0	c0
b0	c1
b1	c2
...	...

T

C	D
c0	d0
c1	d1
c2	d2
...	...

Map

Reduce输入

key	value
(b0, -)	(a0, R)
(b1, -)	(a1, R)
...	...
(b0, c0)	(-, S)
(b0, c1)	(-, S)
...	...
(-, c0)	(d0, T)
(-, c1)	(d1, T)
(-, c2)	(d2, T)
...	...

key	value
(b0, -)	(a0, R)
(b1, -)	(a1, R)
...	...
(b1, c2)	(-, S)
...	...
(-, c0)	(d0, T)
(-, c1)	(d1, T)
(-, c2)	(d2, T)
...	...

Reduce

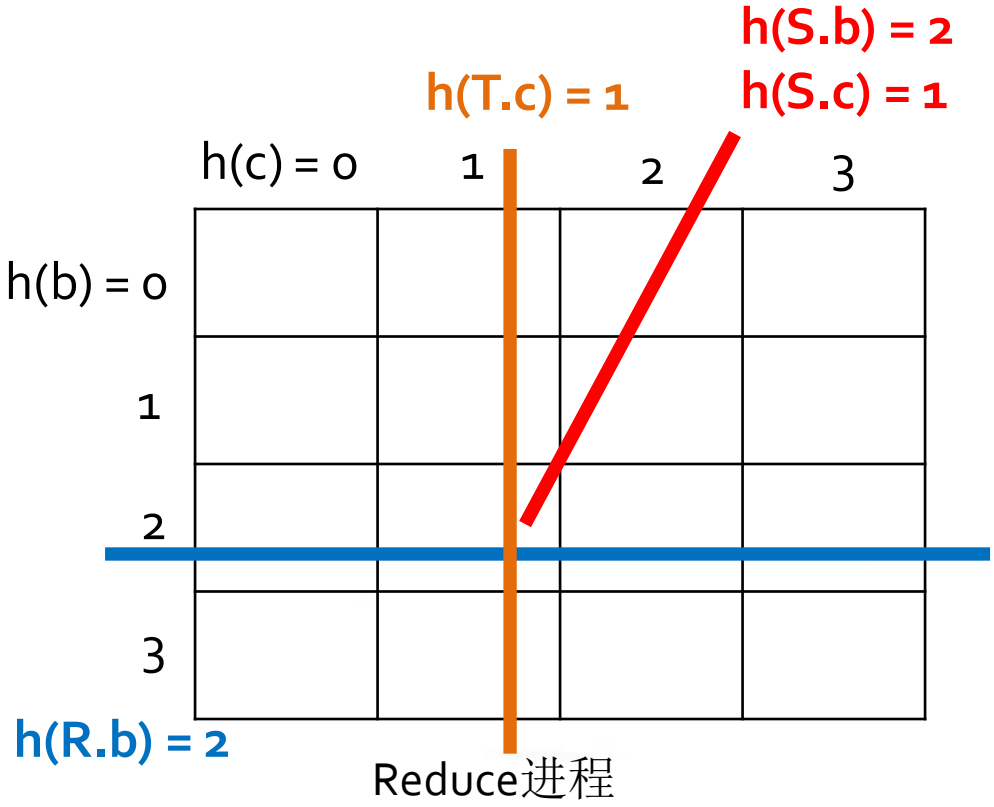
最后输出

A	B	C	D
a0	b0	c0	d0
a0	b0	c1	d1
a1	b1	c2	d2
...	...	...	...

一次处理多个连接

R(A,B)⋈ S(B,C)⋈ T(C,D)

- 令h为取值范围在1, 2, …, *m*的hash函数
  - S(b, c) -> (h(b), h(c))
  - R(a, b) -> (h(b), all)
  - T(c, d) -> (all, h(c))
- 每个Reduce进程计算其接收到元组上的连接



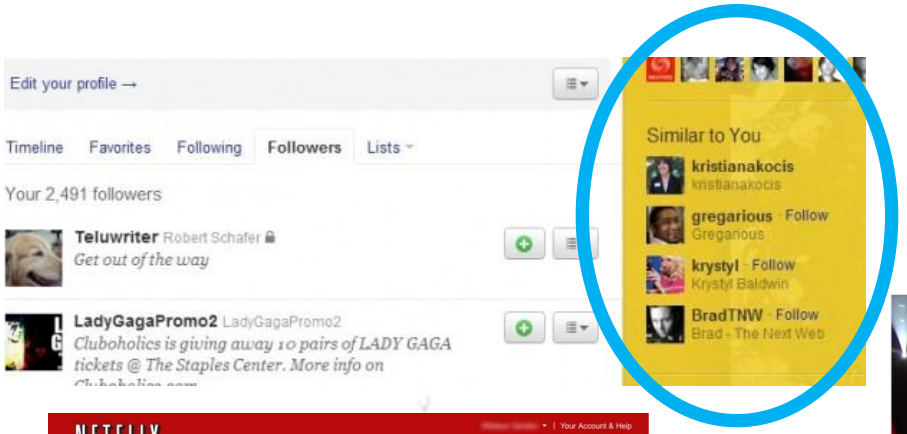
(Reduce进程的数量: $4^2 = 16$)

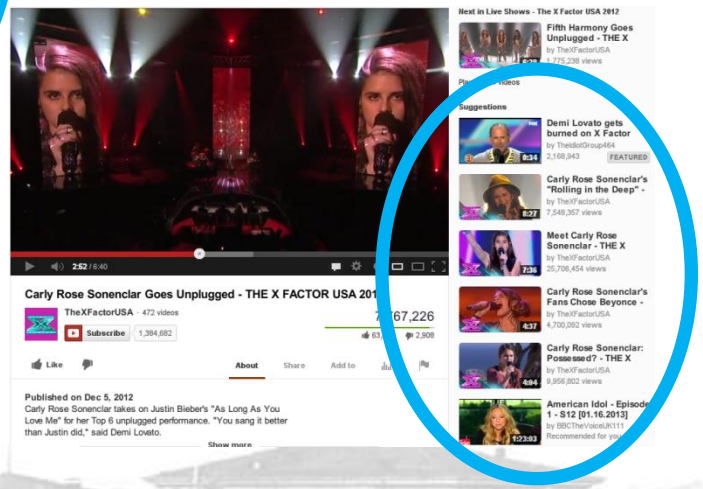
m=4, k=16

相似连接

相似查询为何不够?

- 应用案例
  - 在如twitter的社交网站上寻找相似用户
  - 寻找相似产品, e.g网飞公司的电影推荐和Youtube的相似视频

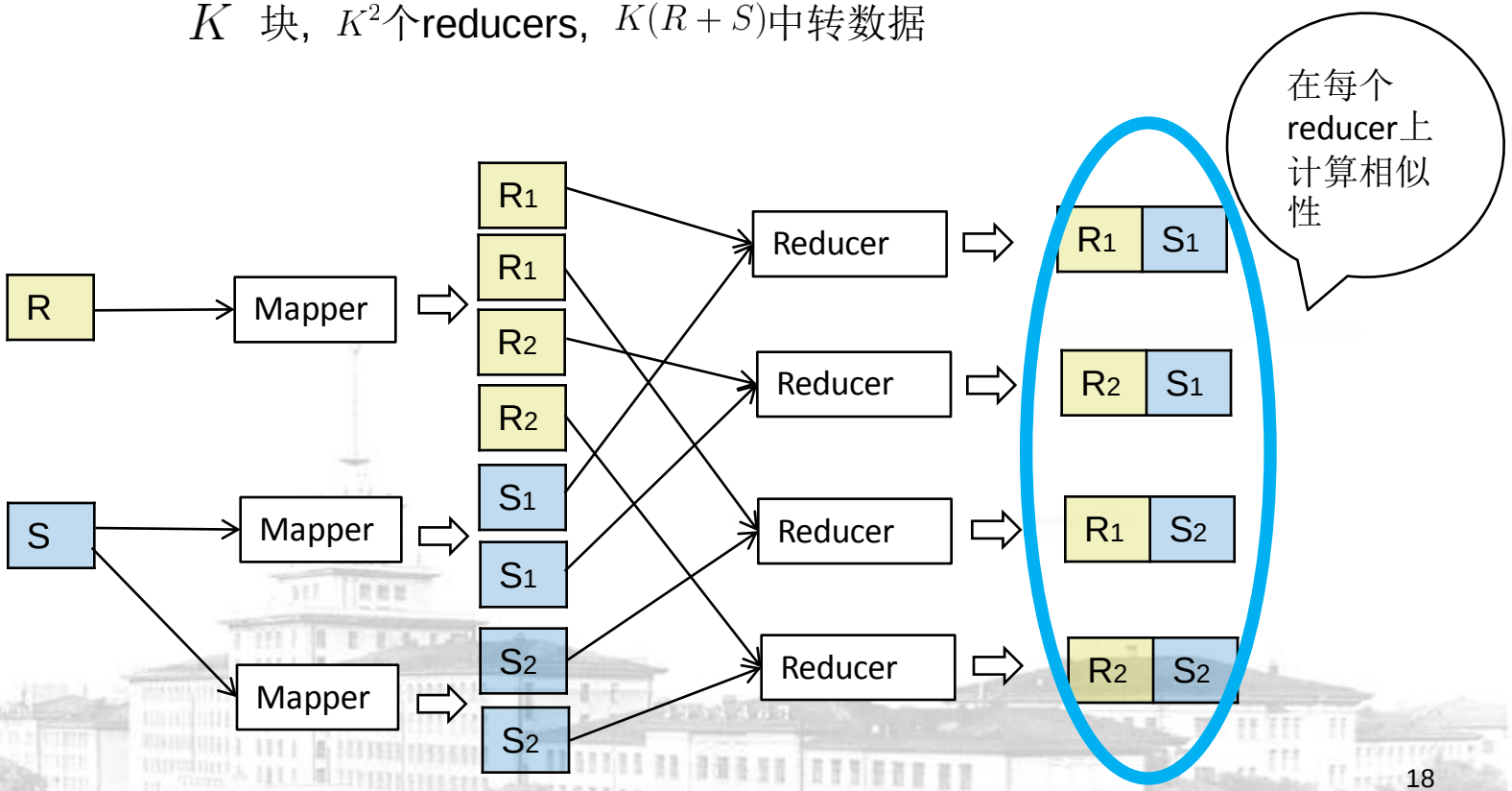




基于MR设计多相似连接的算法

- **问题**：一对来自两个数据集的记录，如果他们的相似性超过一定的程度，它们应该被连接，相似度可以根据特定应用来定义
- **朴素解法**

K 块，K^2个reducers，$K(R+S)$中转数据



用MR进行多重集相似连接的算法

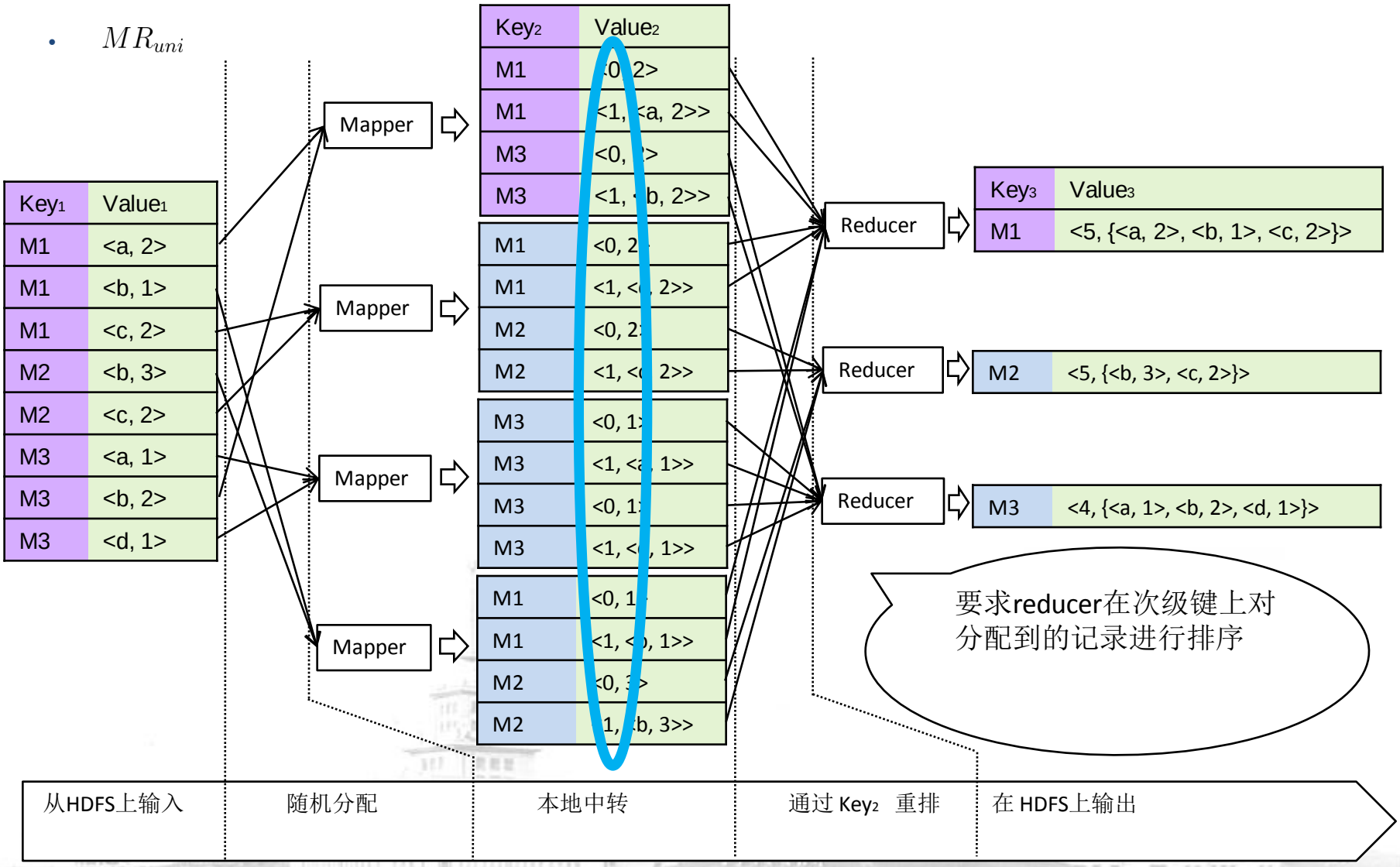
- 多元相似, e.g., $M_1 = \{< a,2 >, < b,1 >, < c,3 >\}, M_2 = \{< a,1 >, < c,2 >, < d,2 >\}$
  - Jaccard 相似性 $\frac{|M_1 \cap M_2|}{|M_1 \cup M_2|} = \frac{|<a,min(2,1)>,<c,min(3,2)>|}{|<a,max(2,1)>,<b,1>,<c,max(3,2)>,<d,2>|} = \frac{1+2}{2+1+3+2} = \frac{3}{8}$
  - Cosine 相似性 $\frac{M_1 \cap M_2}{\sqrt{|M_1|+|M_2|}} = \frac{1+2}{\sqrt{(2+1+3)+(1+2+2)}} = \frac{3}{\sqrt{11}}$
  - Dice相似性 $2 \times \frac{|M_1 \cap M_2|}{|M_1|+|M_2|} = \frac{2 \times (1+2)}{(2+1+3)+(1+2+2)} = \frac{6}{11}$

  - 常见计算
  - **单元**函数, e.g., $|M|$
  - 合取函数, e.g., $|M_1 \cap M_2|$
  - 析取函数, e.g., $|M_1 \cup M_2| = |M_1| + |M_2| - |M_1 \cap M_2|$

- 自相似连接的可扩展性
  - **第一个** MapReduce 工作, MR_{uni}计算单元函数值
  - **其他** MapReduce 工作, MR_{sim}计算合取函数值和相似性

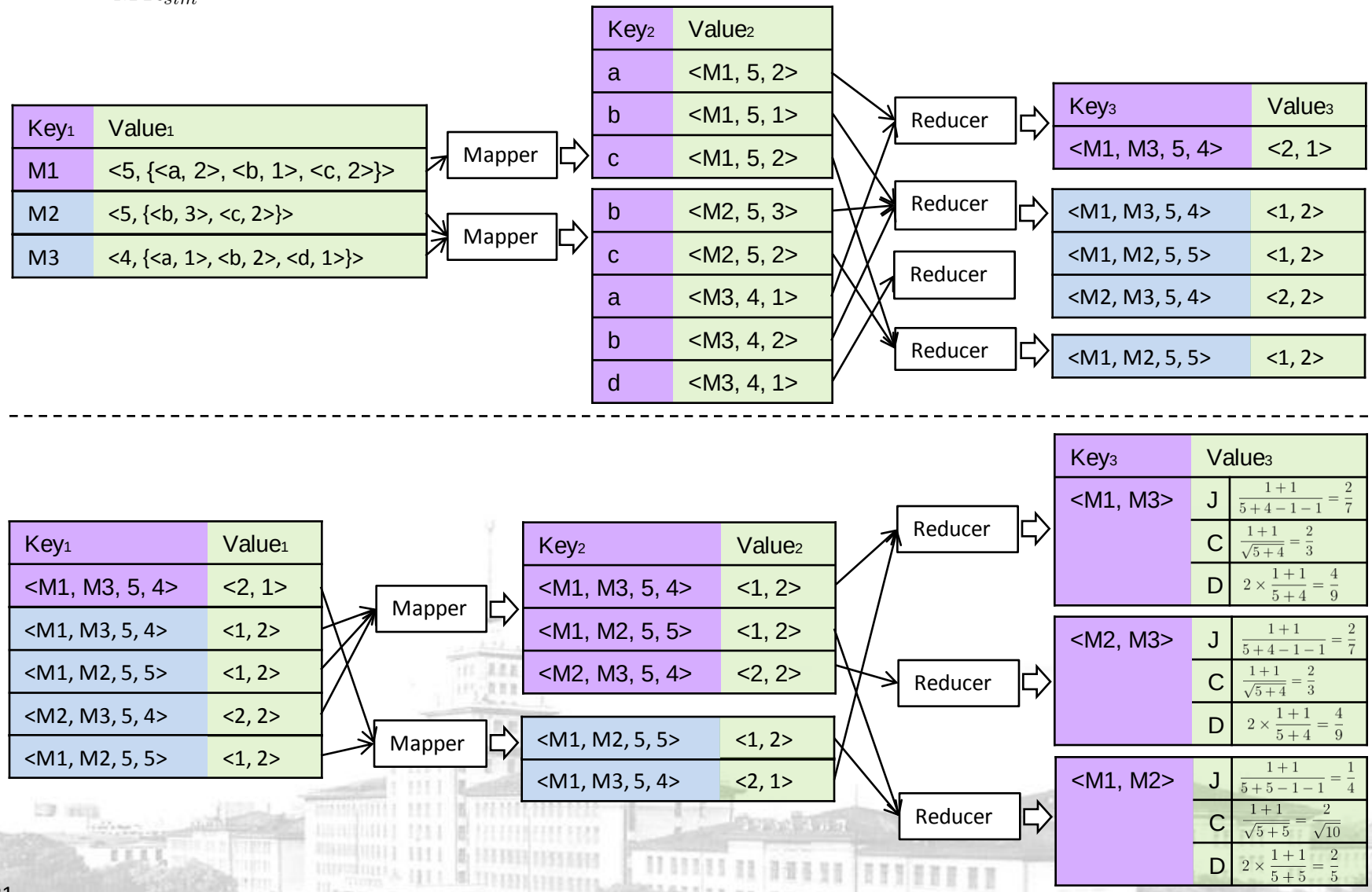
多重集合相似连接算法

- MR_{uni}



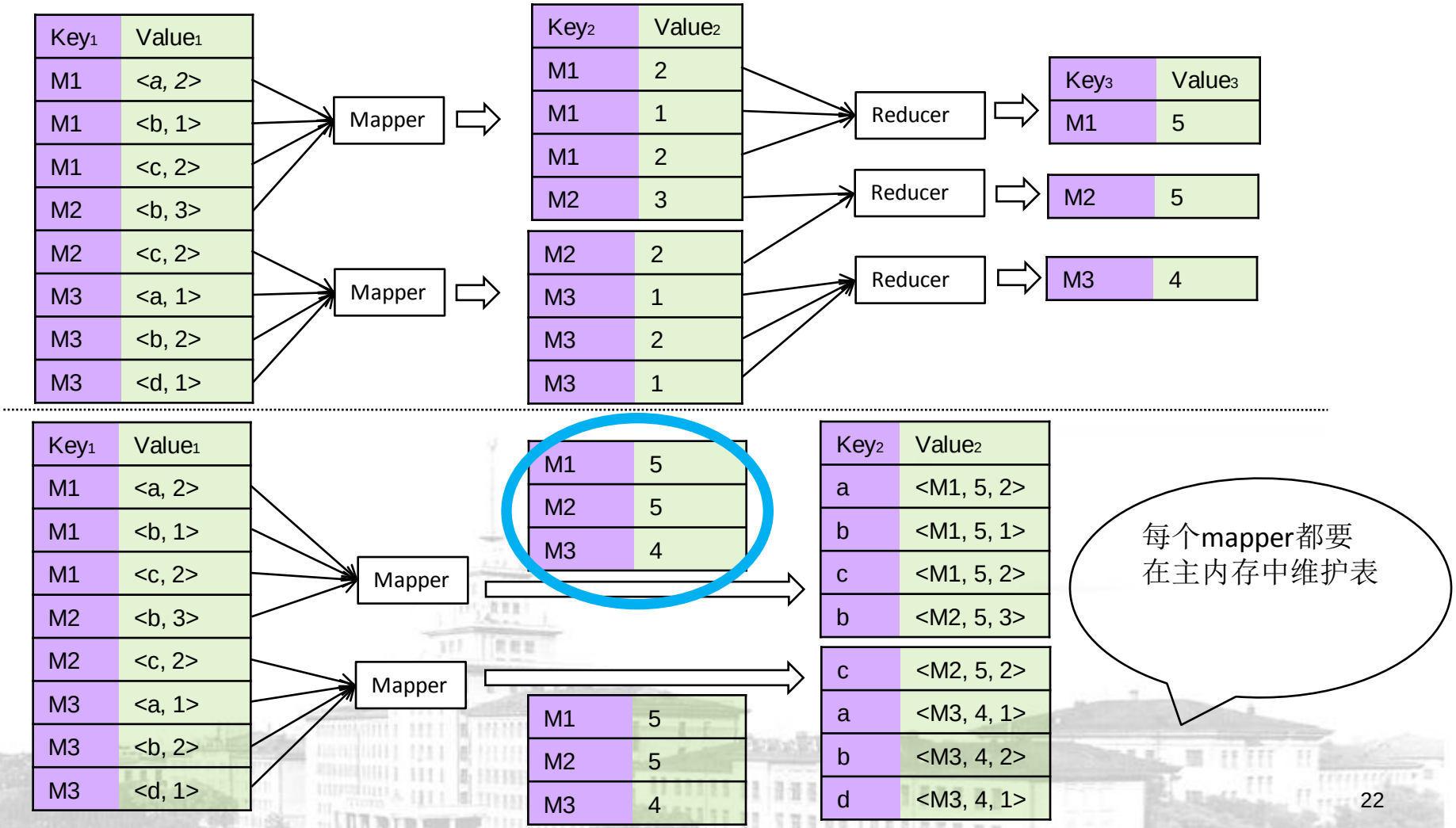
多重集合相似连接的算法

- MR_{sim}



多集合相似连接的算法

- MR_{uni} 的变型
- 通过两个任务查找单元函数值

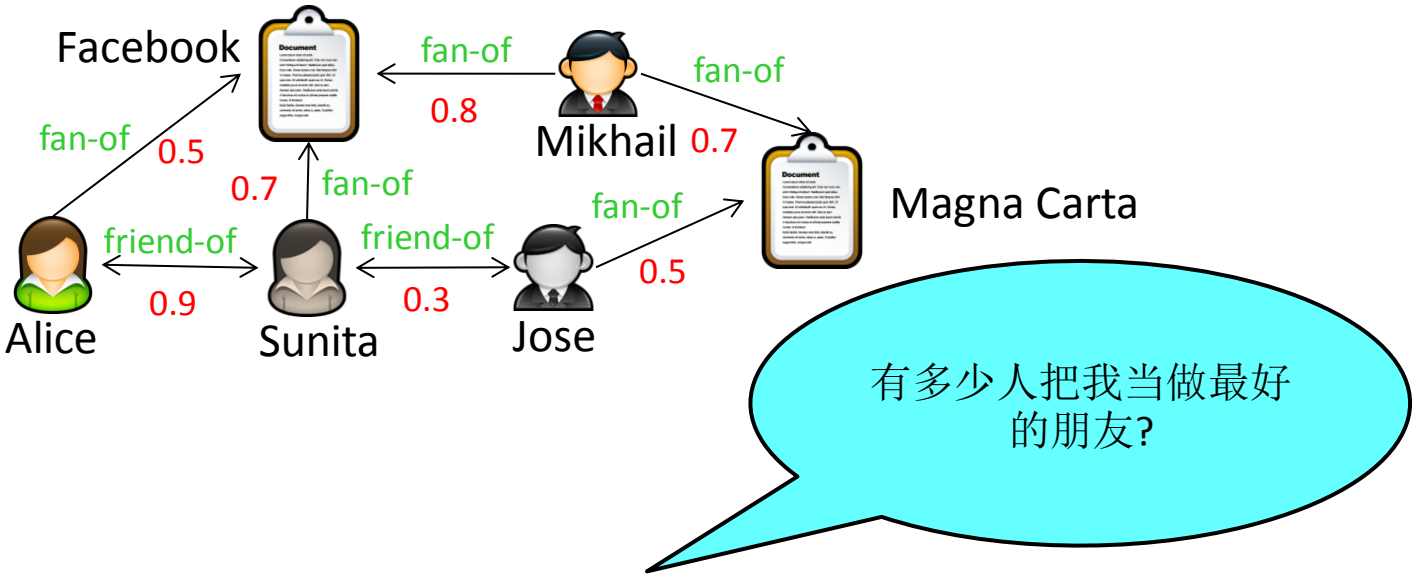


本讲内容

8.1 连接算法

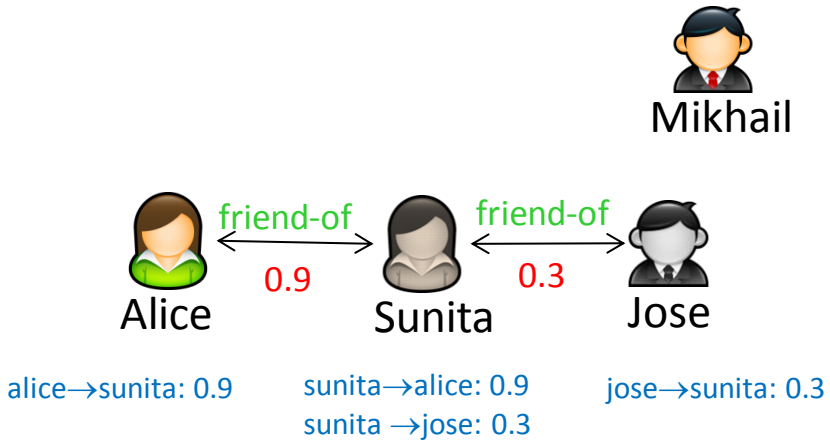
8.2 图算法

图的计算模型



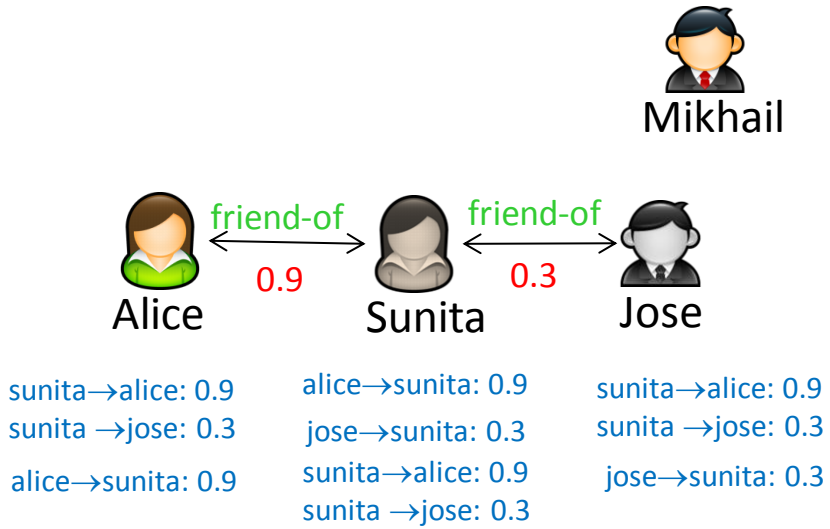
- 例子: 我是我朋友的最好朋友么？
  - 步骤 #1: 去掉无关的边和结点

图的计算模型



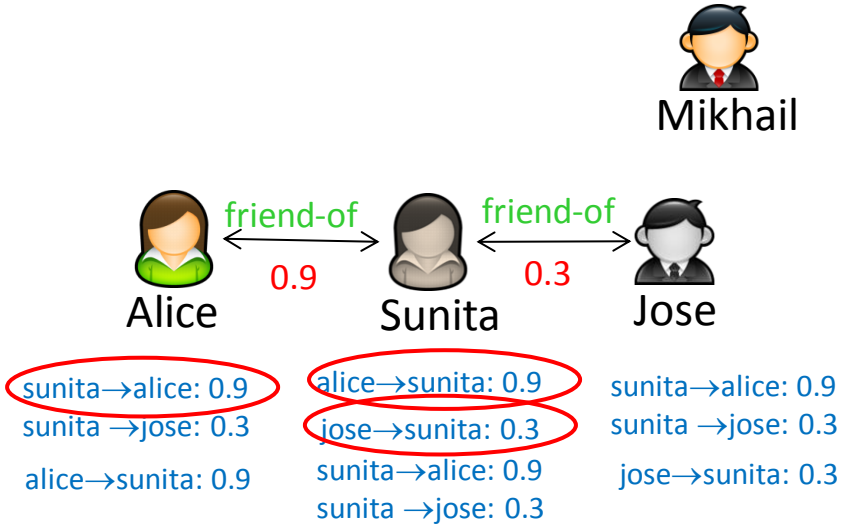
- 例子: 我是我朋友的最好朋友么?
  - 步骤 #1: 去掉无关的边和结点
  - 步骤 #2: 用朋友列表标记每个结点
  - 步骤 #3: 沿着每条边下推标签

图的计算模型



- 例子: 我是我朋友的最好朋友么？
  - 步骤 #1: 去掉无关的边和结点
  - 步骤 #2: 用朋友列表标记每个结点
  - 步骤 #3: 沿着每条边下推标签

图的计算模型



- 例子: 我是我朋友的最好朋友么？
  - 步骤 #1: 去掉无关的边和结点
  - 步骤 #2: 用朋友列表标记每个结点
  - 步骤 #3: 沿着每条边下推标签
  - 步骤 #4: 为每个结点确定结果

可否用MapReduce实现？

```
map(key: node, value: [<otherNode, relType, strength>])
{


}
reduce(key: _________, values: list of __________)
{


}
```

- 使用邻接表表示?

可否用MapReduce实现？

```
map(key: node, value: <otherNode, relType, strength>)
{


}
reduce(key: _________, values: list of __________)
{


}
```

- 使用单边数据表示?

一个实际用例

- 一个在社交网络中常见的变种：谁是我多个朋友的朋友？
- 朋友推荐!
  - 这些人可能是也是我的朋友!

更一般的情况…

- 假设我们希望超越直接的朋友关系
  - 例如: 有多少我朋友的朋友把我当做他们最好朋友(距离为2的邻居)的最好朋友?
  - 我们需要做什么?
- 距离k>2的情况如何?
- 为了计算答案，我们需要运行多轮MapReduce!

迭代 MapReduce

- 基本模型:

```
从输入路径拷贝文件 → dir 1
(可选: 进行预处理)

while (!结束条件) {
  map 从dir 1
  reduce 到dir 2
  move 文件: 从dir 2 → dir1
}

(可选: 后处理)
从dir 2 → 输出路径移动文件
```

- 注意reduce的输出必须和map的输入兼容!
  - 如果我们将mapper或者reducer中的一些信息过滤掉，会发生什么？

图算法和MapReduce

- 一个中心式算法通常每次扫描树或者图中的一个项 (仅有一个游标)
  - 深度优先广度优先

- 许多图的算法需要多个map/reduce阶段
  - 一些时候需要迭代MapReduce, 另外一些时候需要map/reduce链

图算法和MapReduce

- 假设我们需要:
  - 为图中的每个点计算函数
  - 使用至多k-hop以外顶点中的数据
- 我们可以这样做:
  - 将信息延着边推送
    - “像结点一样思考"
  - 在每个结点上完成计算
- 可能需要多于一个MapReduce阶段
  - 迭代MapReduce: 阶段i的输出 → 阶段i+1的输入

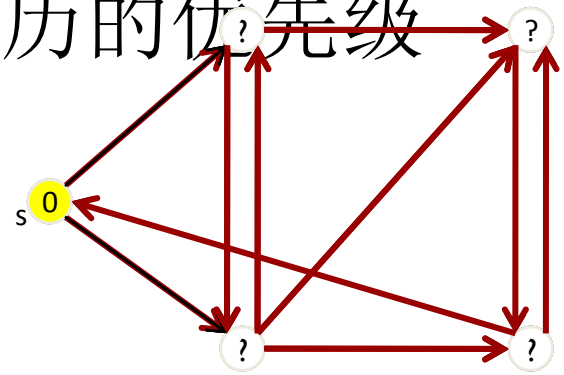
基于路径的算法

- 目标是计算结点间关于路径的信息
  - 边的标记包括代价, 距离, 或者相似性

- 这类问题的例子:
  - 单源最短路径
  - 最小生成树
  - Steiner树 (连接给定集合的最小代价树)
  - 拓扑排序

单源最短路径: 如何并行化?

- Dijkstra算法每次沿着一个中间结点遍历图，基于总路径长度 定义遍历的优先级
  - 这里无需并行化!



- 直观的看，我们需要从源辐射的出来的点，每次增加一跳
  - 在下一轮开始之前，向外的每一步可以并行的做

SSSP: 重温递归定义

```
bestDistanceAndPath(v) {
  if (v == source) then {
    return <distance 0, path [v]>
  } else {
    find argmin_u (bestDistanceAndPath[u] + dist[u,v])
    return <bestDistanceAndPath[u] + dist[u,v], path[u] + v>
  }
}
```

- Dijkstra算法仔细考察每个u，从而可以安全删除确定顶点
- 我们对每个v考察所有潜在的u
  - 通过保存u的前沿集合迭代计算(距离源i条边)

SSSP: MapReduce正规化

- 初始化:
  - 对于每个结点node ID → <∞, -, {<succ-node-ID,edge-cost>}>
- map:
  - 考虑结点ID → <dist, next, {<succ-node-ID,edge-cost>}>
  - 对于每个succ-node-ID:
    - 传送succ-node ID → {<node ID, distance+edge-cost>}
  - 传送ID → distance,{<succ-node-ID,edge-cost>}
- reduce:
  - distance := 前驱的最小代价; next := 具有最小代价的前驱.
  - 传送ID → <distance, next, {<succ-node-ID,edge-cost>}>
- 重复，直到不发生变化
- 后处理: 去掉邻接表

从源到结点ID的最短路径长度为∞...

... 路径上的下一个结点

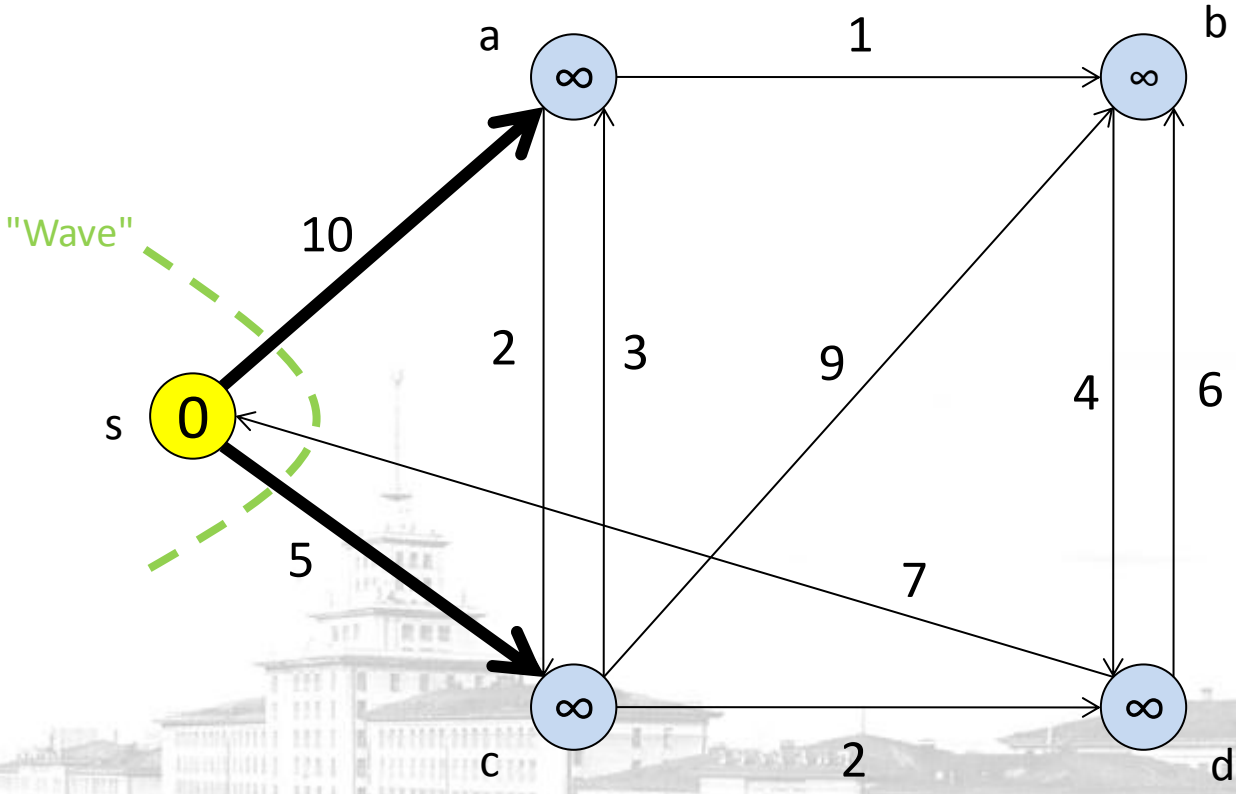
结点ID的邻接表

这是从源到succ-node-ID新发现的路径(无需最短)

为何必要?

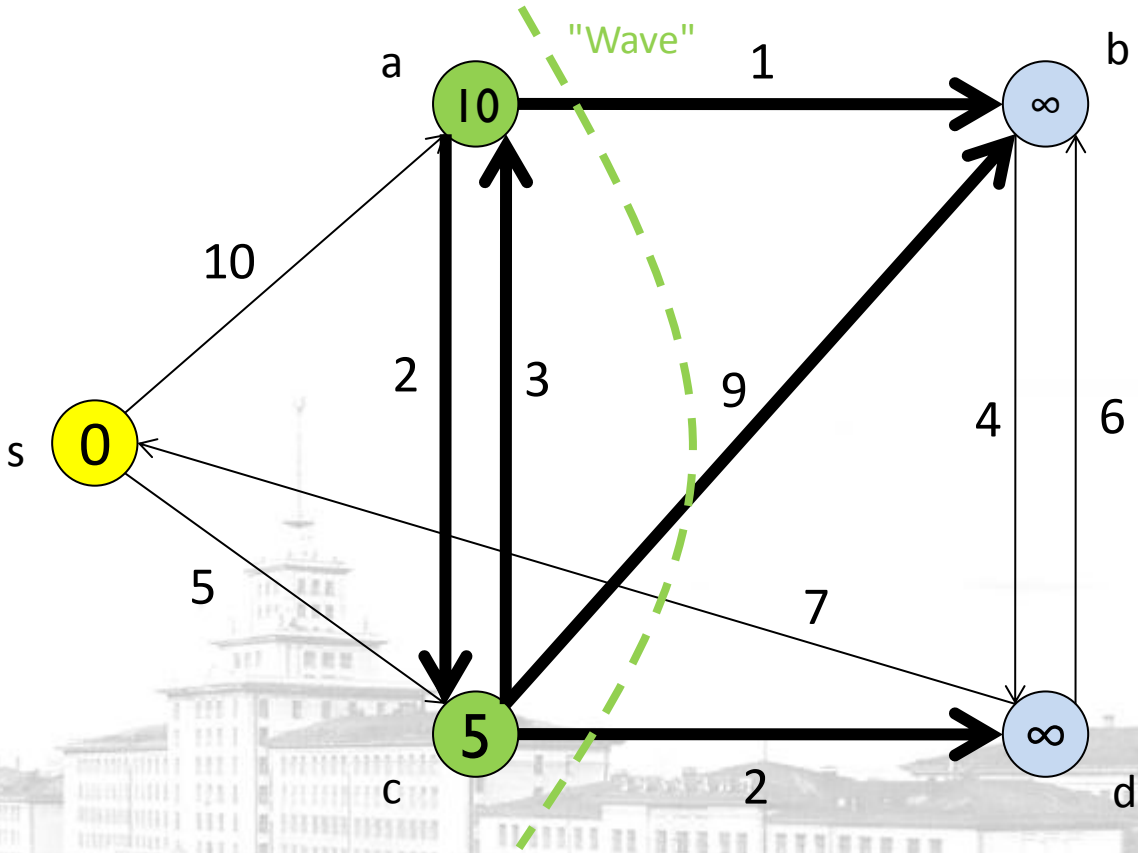
阶段0: 基本情况

mapper:	(a,<s,10>) (c,<s,5>)
reducer:	(a,<10, ...>) (c,<5, ...>)



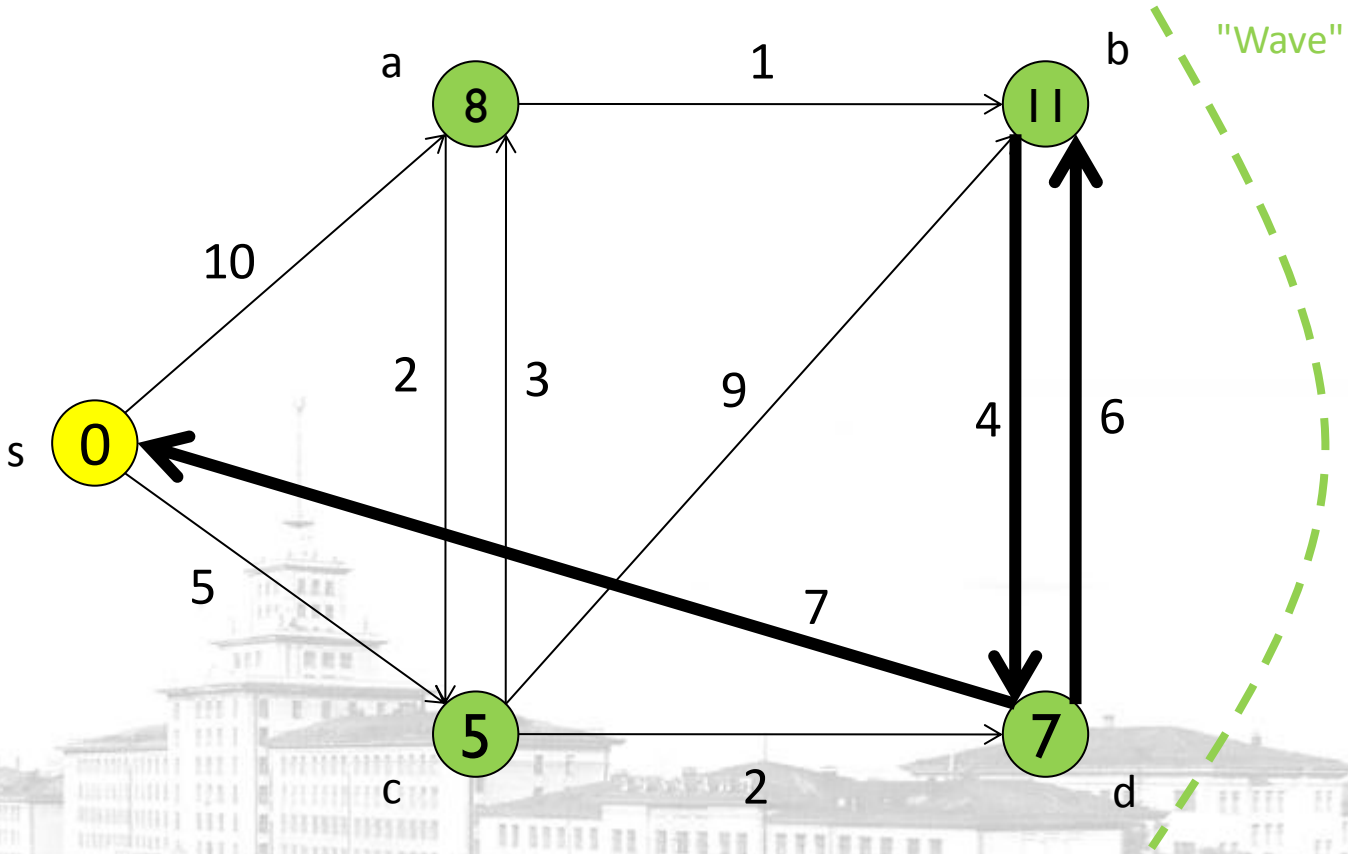
阶段1

mapper:	(a,<s,10>) (c,<s,5>) (a,<c,8>) (c,<a,9>) (b,<a,11>) (b,<c,14>) (d,<c,7>)
reducer:	(a,<8, ...>) (c,<5, ...>) (b,<11, ...>) (d,<7, ...>)



阶段2

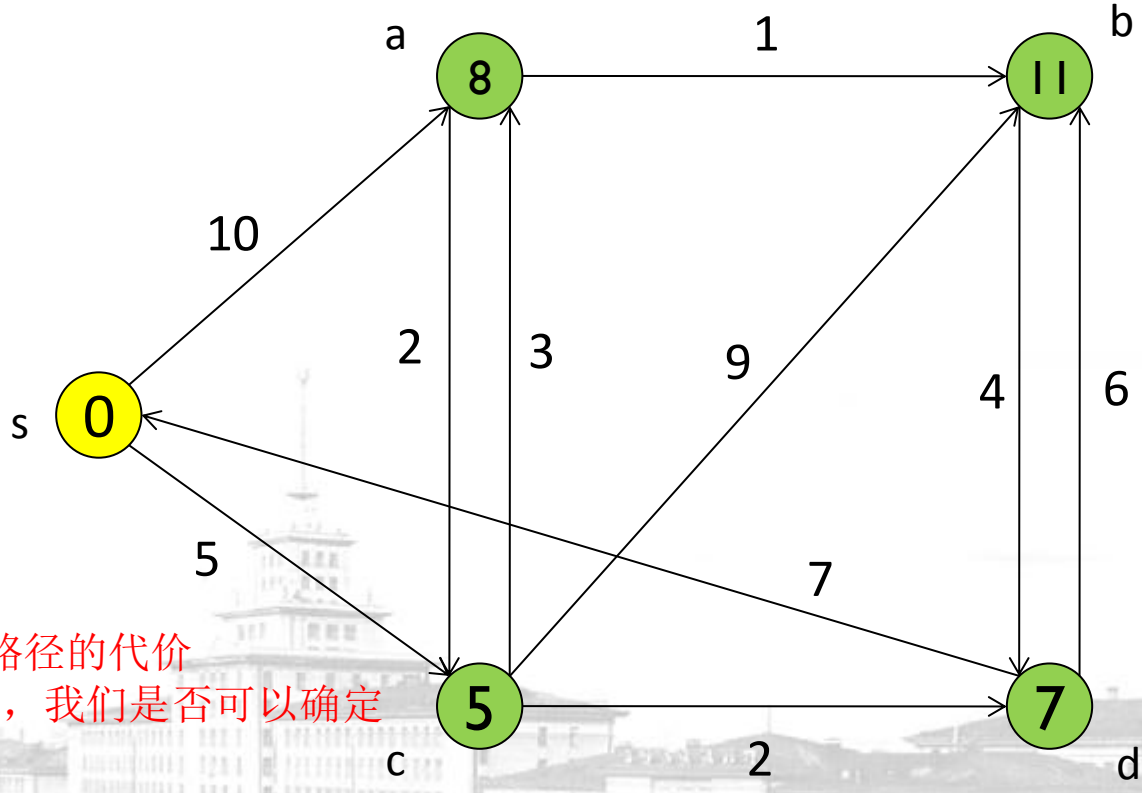
mapper: (a,<s,10>) (c,<s,5>) (a,<c,8>) (c,<a,9>) (b,<a,11>) (b,<c,14>) (d,<c,7>) (b,<d,13>) (d,<b,15>)
reducer: (a,<8>) (c,<5>) (b,<11>) (d,<7>)



阶段3

不变化! 收敛!

mapper:	(a,<s,10>) (c,<s,5>) (a,<c,8>) (c,<a,9>) (b,<a,11>) (b,<c,14>) (d,<c,7>) (b,<d,13>) (d,<b,15>)
reducer:	(a,<8>) (c,<5>) (b,<11>) (d,<7>)



SSSP摘要

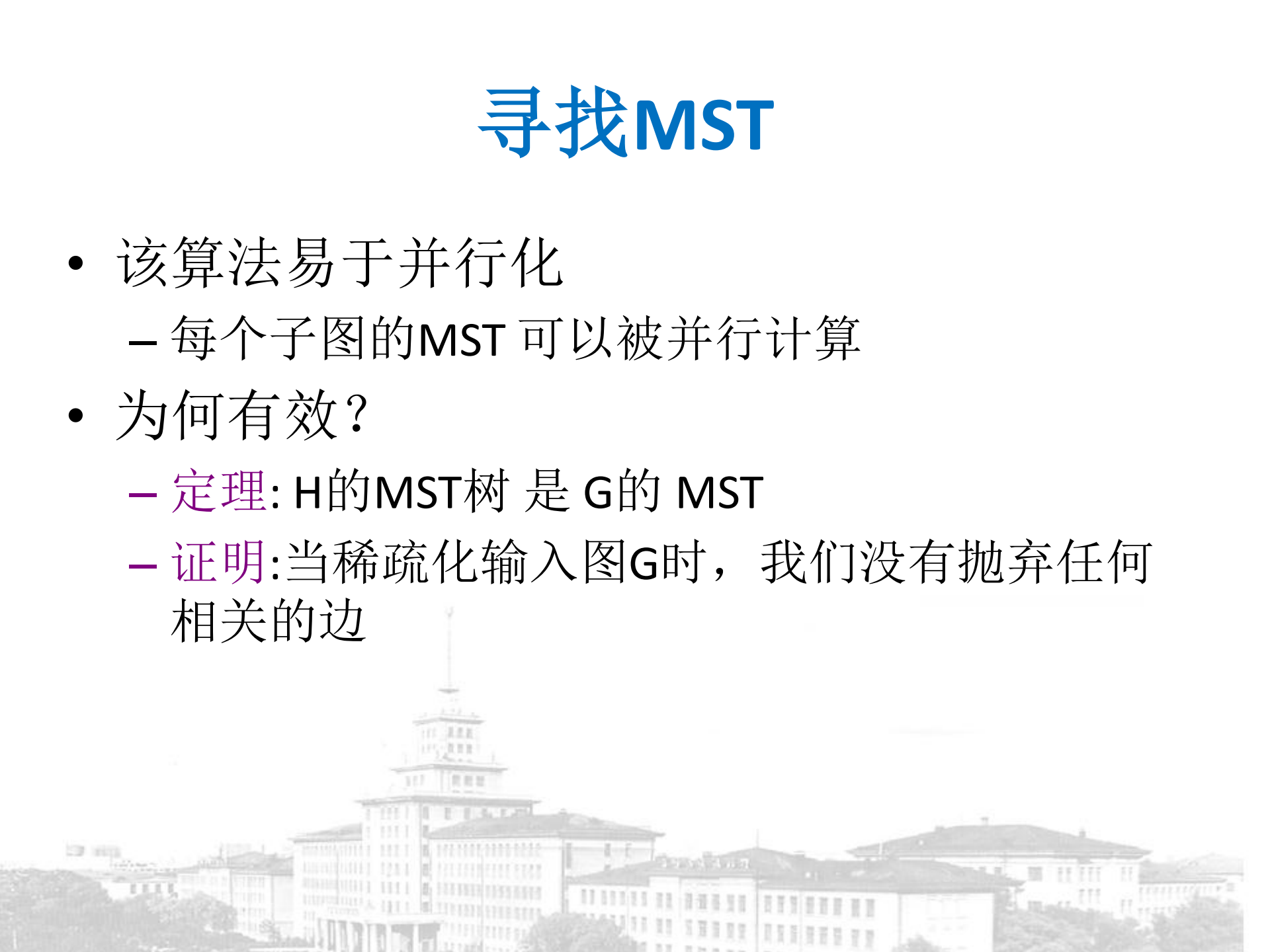
- 基于路径的算法经常涉及到迭代map/reduce
- 经常把迭代正规化为”waves”或者”阶段”,类似BFS
  - 允许并行化
  - 需要收敛检测
- 例子：SSSP
  - Dijkstra算法难以并行化
  - 但是我们可以让其通过“wave” 方法运行

计算MST

- 问题:找到一个稠密图的最小生成树
- 算法
  - 随机划分顶点为k部分
  - 对于每一对顶点集,找到这两个集合导出二分子图的MST
  - 对这样求出的所有MST的边取并集，生成图H
  - 计算H的MST

寻找MST

- 该算法易于并行化
  - 每个子图的MST 可以被并行计算
- 为何有效？
  - 定理: H的MST树 是 G的 MST
  - 证明:当稀疏化输入图G时，我们没有抛弃任何相关的边



致谢

- 本讲义部分内容来自于Rui Zhang, A. Haeberlen和Z. Ives的讲义