中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

SSE316：云计算技术
Cloud Computing Technology

陈壮彬

软件工程学院

https://zbchern.github.io/sse316.html

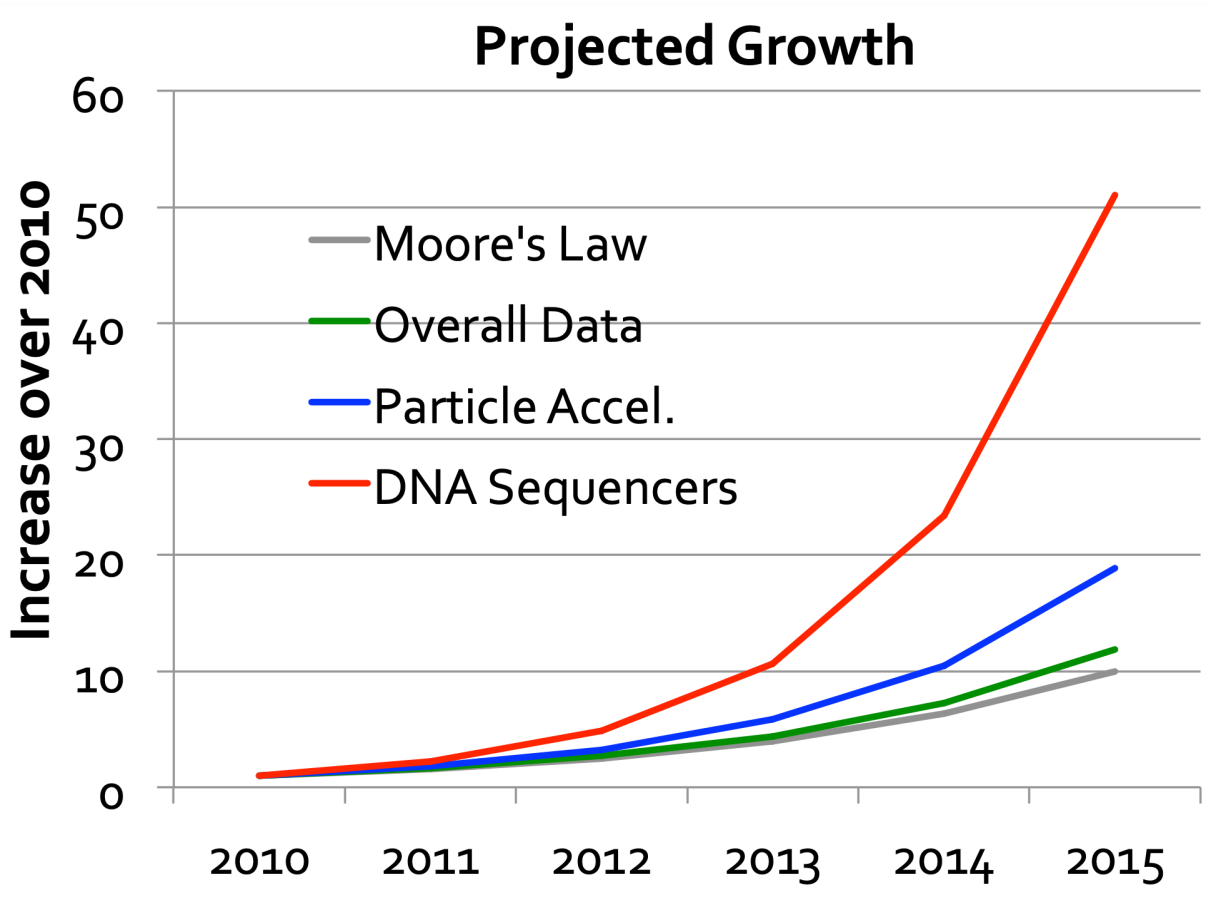
云资源管理

- 基于抽样的大数据应用
- 基于数据感知的资源调度
  - 数据局部性
  - 数据传输的均衡性

云资源管理

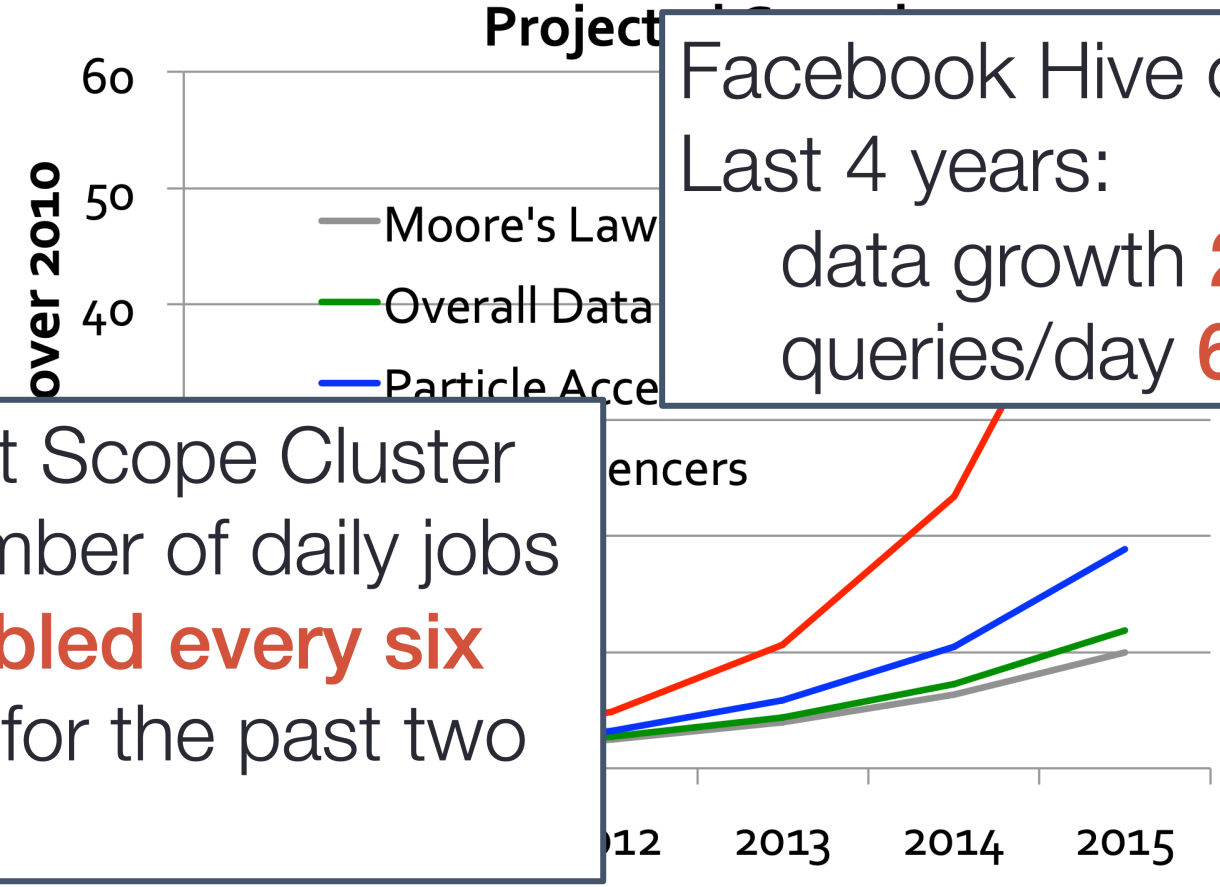
- 基于抽样的大数据应用
- 基于数据感知的资源调度
  - 数据局部性
  - 数据传输的均衡性

数据增长趋势（1）



Data grows faster than Moore’ s Law

数据增长趋势（2）



Facebook Hive cluster
Last 4 years:
data growth **2500x**
queries/day **60x**

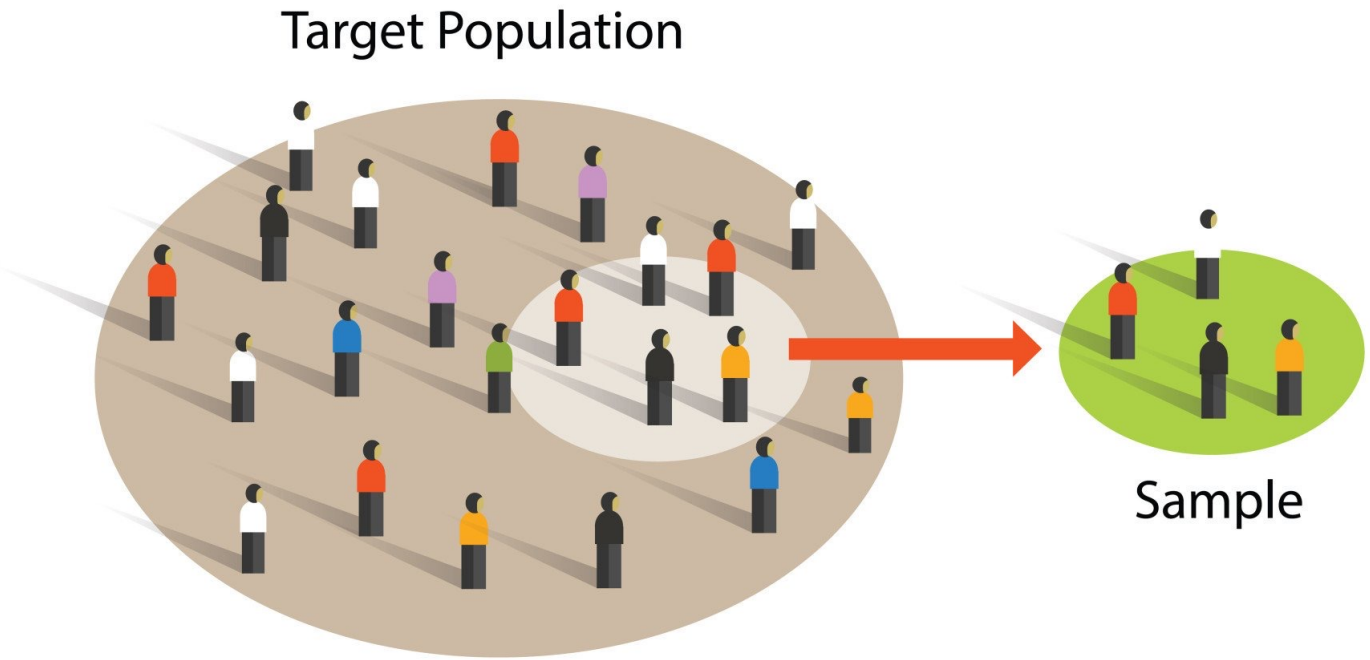
Microsoft Scope Cluster
"The number of daily jobs has **doubled every six months** for the past two years."

Data grows faster than Moore's Law

Based on the data in 2014

基于数据抽样的大数据应用

- 通过计算数据的一个子集快速得到近似的结果

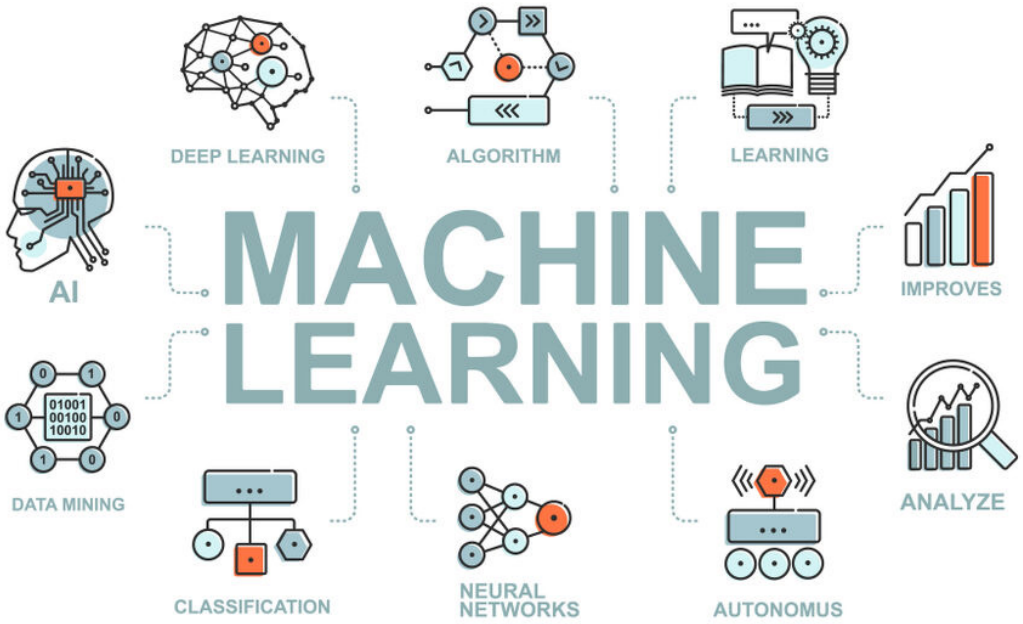


近似查询处理

- 近似查询处理（Approximate Query Processing, AQP）
  - ✓产品分析员可以使用AQP系统，根据点击率样本快速决定是否需要更改广告活动

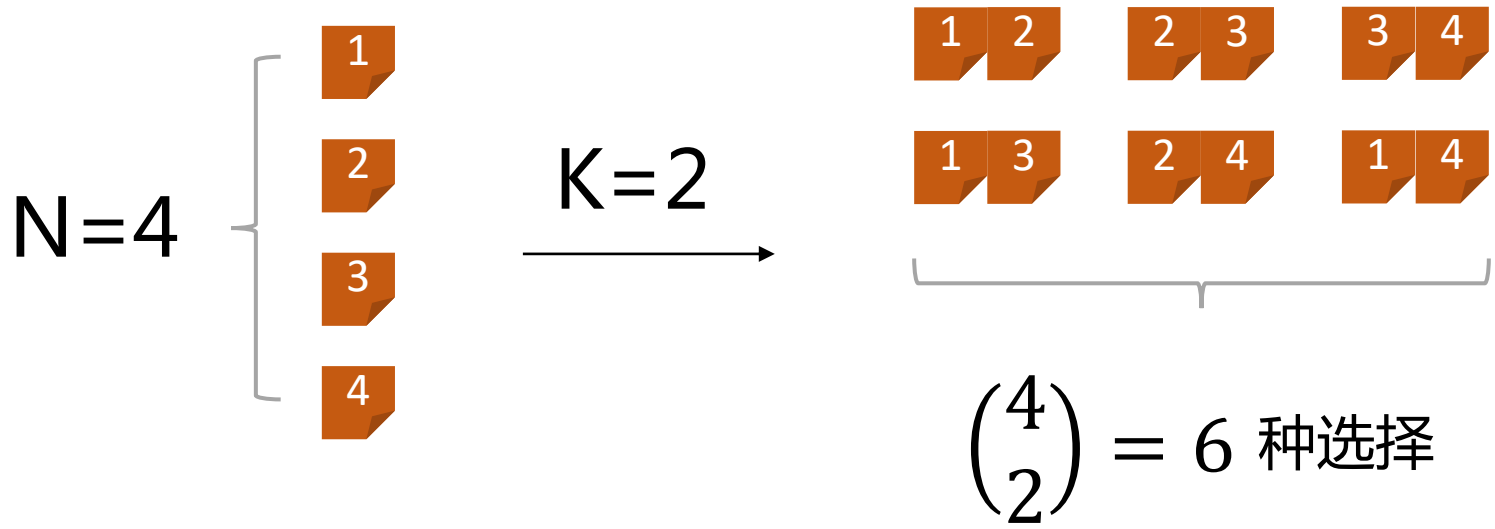
机器学习应用

- 机器学习应用（Machine learning applications）
  - ✓提出算法的随机版本，比如随机梯度下降法
  - ✓使用小的随机数据样本，即使对于大型数据集也能提供统计上有效的结果



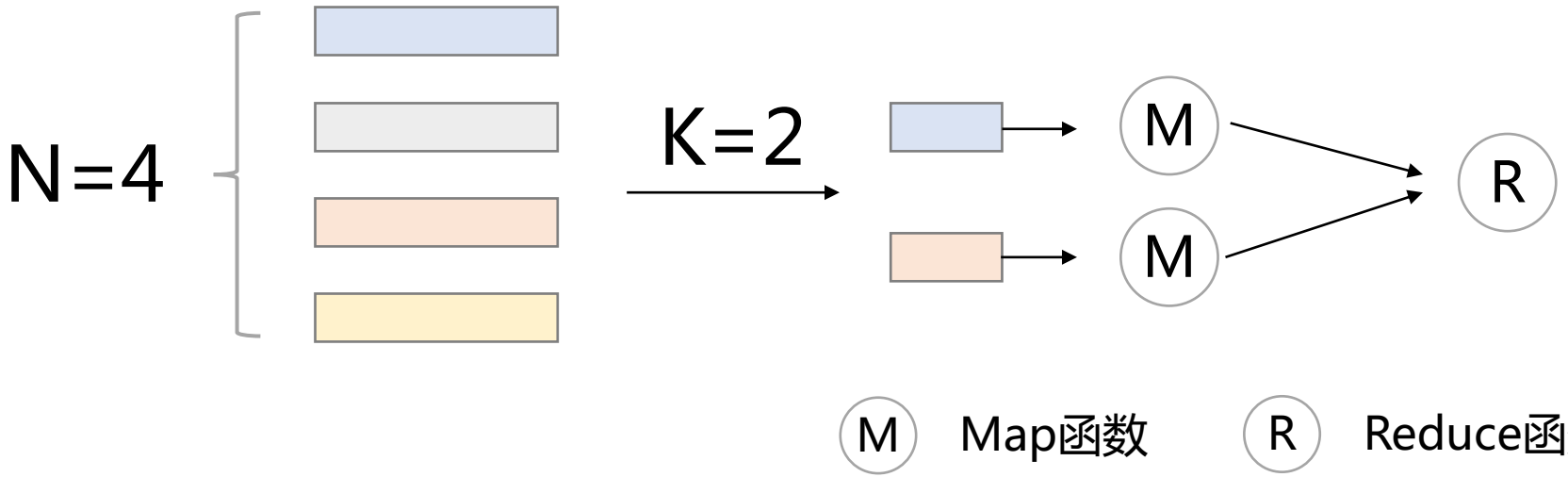
数据块选择（1）

数据块选择（2）

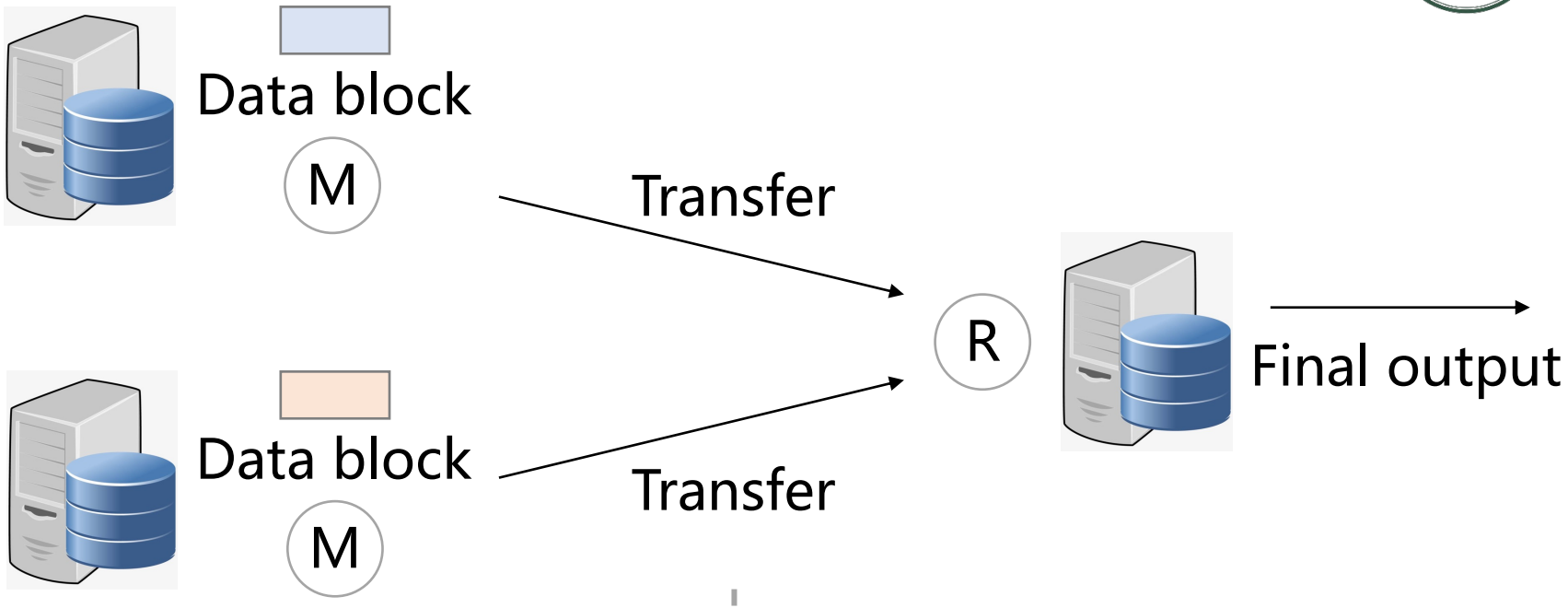


$\binom{4}{2} = 6$ 种选择

例子



如何让任务运行的更快？



上游任务

增加数据局部性

下游任务

加快数据传输

数据感知的调度

The Power of Choice in Data-Aware Cluster Scheduling

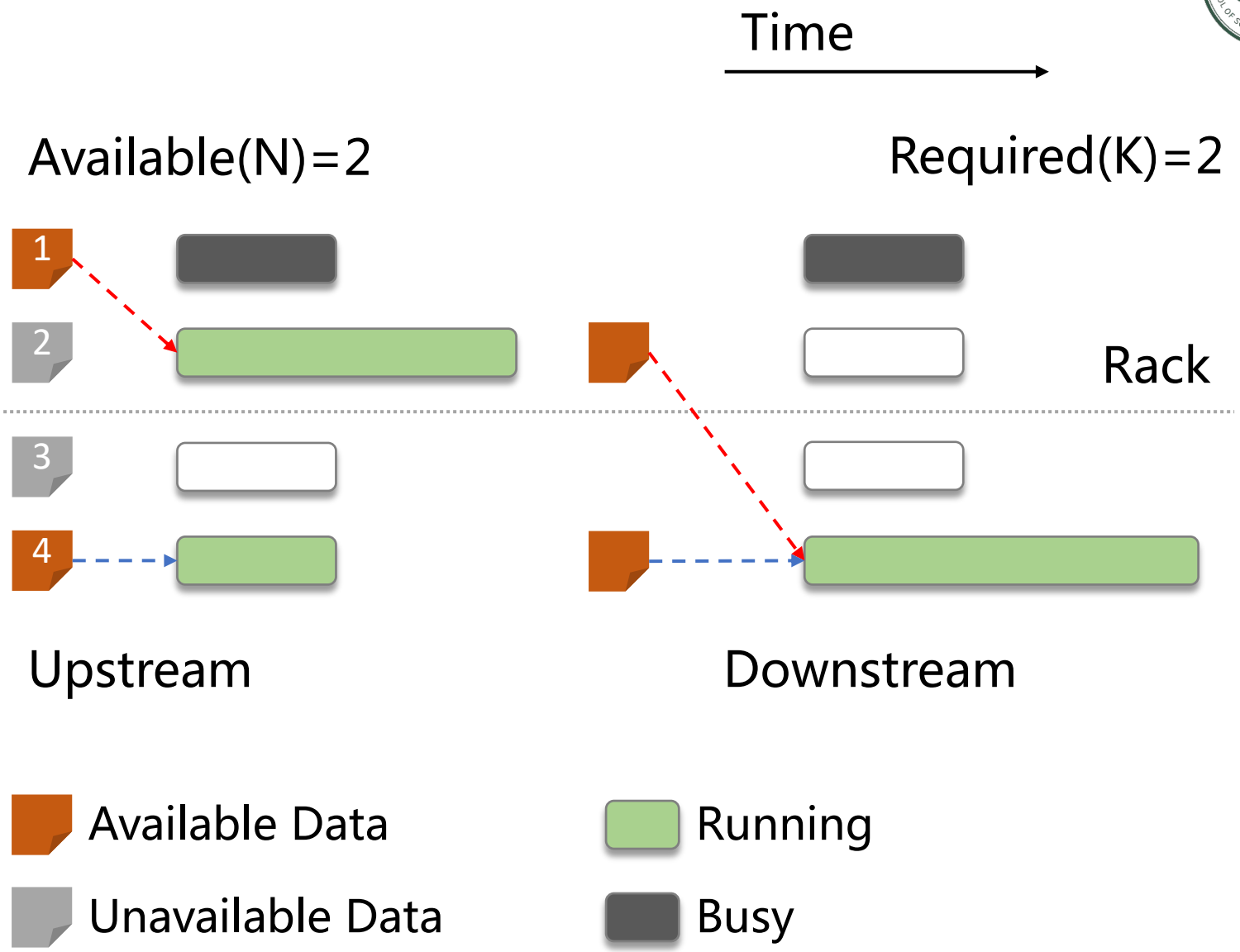
Shivaram Venkataraman[1], Aurojit Panda[1], Ganesh Ananthanarayanan[2], Michael J. Franklin[1], Ion Stoica[1]
[1] *UC Berkeley* [2] *Microsoft Research*

OSDI 2014

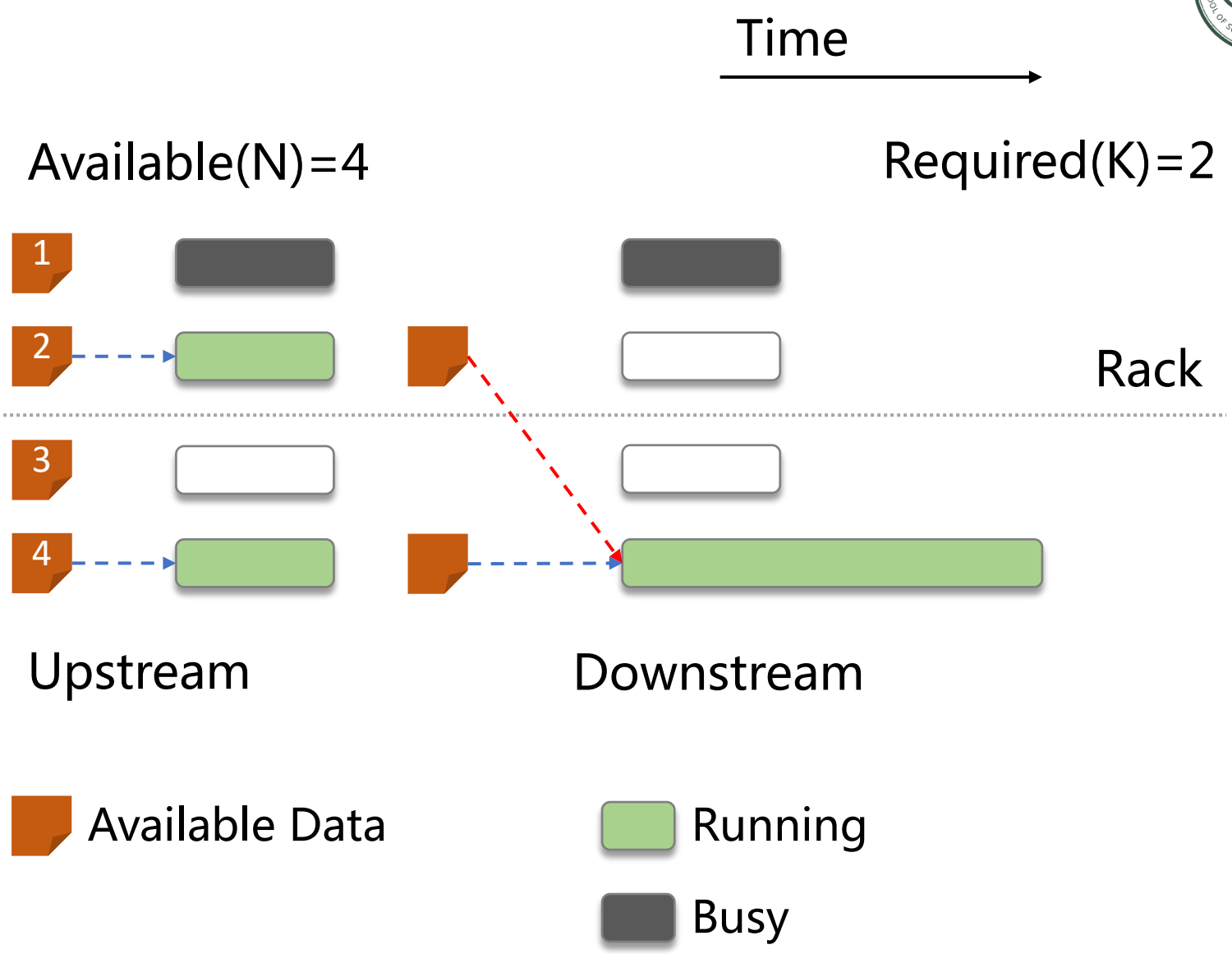
云资源管理

- 基于抽样的大数据应用
- 基于数据感知的资源调度
  - 数据局部性
  - 数据传输的均衡性

随机选择数据块


Time
Available(N)=2
Required(K)=2
1
2
3
4
Rack
Upstream
Downstream
Available Data
Unavailable Data
Running
Busy

数据感知的选择


Time
Available(N)=4
Required(K)=2
1
2
3
4
Rack
Upstream
Downstream
Available Data
Running
Busy

算法框架

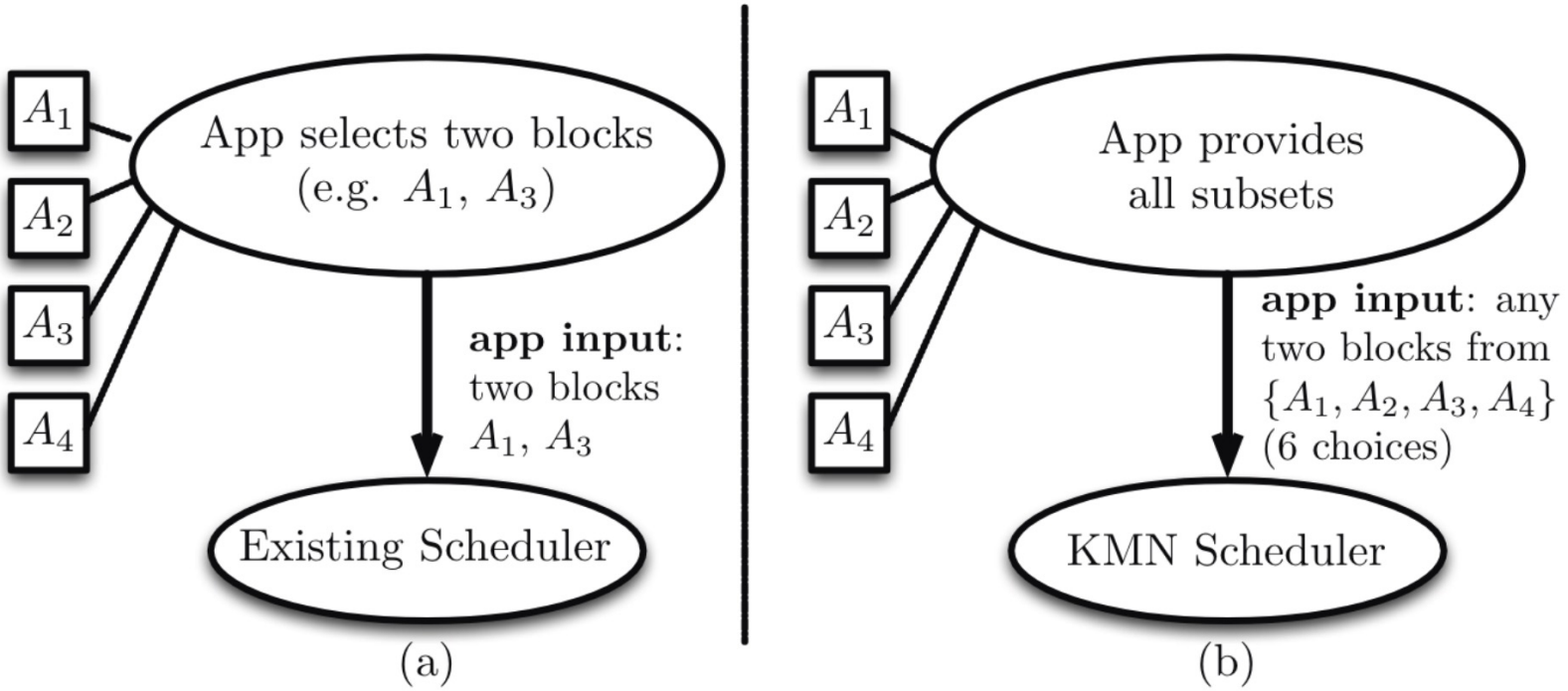


Figure 1: **"Late binding" allows applications to specify more inputs than tasks and schedulers dynamically choose task inputs at execution time.**

数据局部性的概率（1）

- 假设 u 为集群的利用率，若集群中的节点有 s 个slots，则该节点的所有 slots 被占用（即不可用）的概率为

$$u^s$$

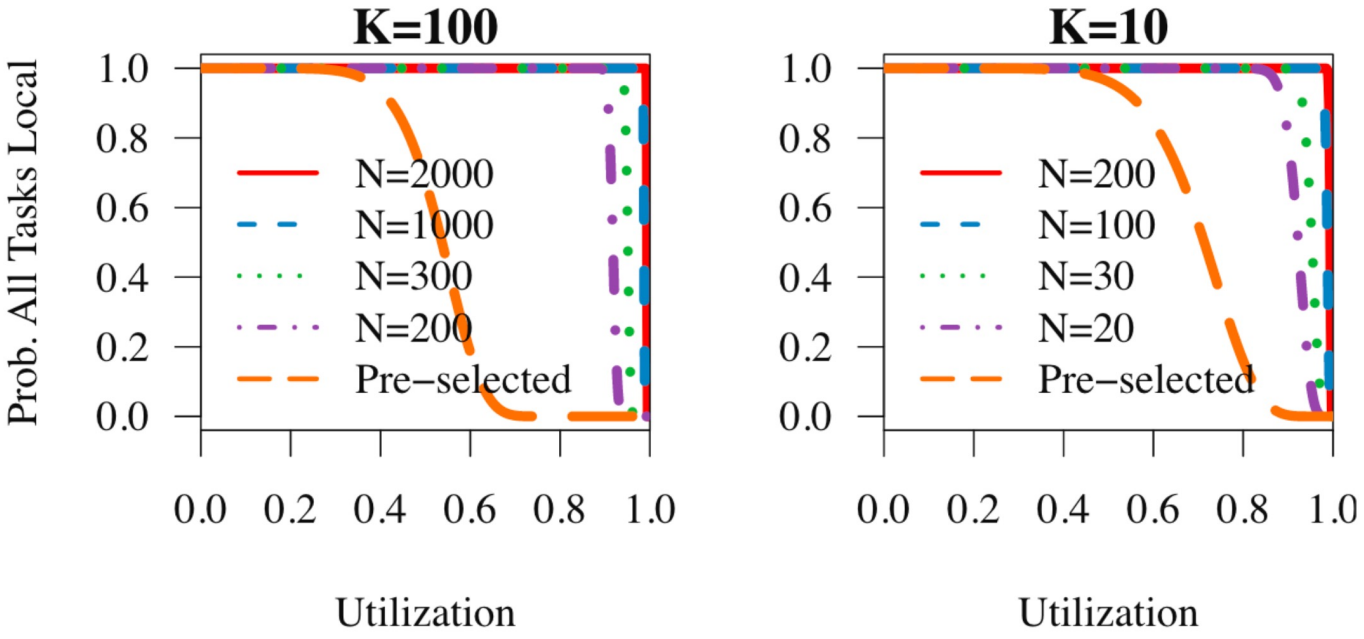
- 因此，对于该节点上的数据块，数据局部性的概率为

$$p_t = 1 - u^s$$

数据局部性的概率（2）

- 对于这样的集群，从 N 个任务中选出的 K 个任务均获得数据局部性的概率由成功概率为 p_t 的二项式 CDF 函数给出：

$$\sum_{i=0}^{K-1} \binom{N}{i} p_t^i (1-p_t)^{N-i}$$



即便集群的占用率为90%，实现数据局部性的概率仍然很高。

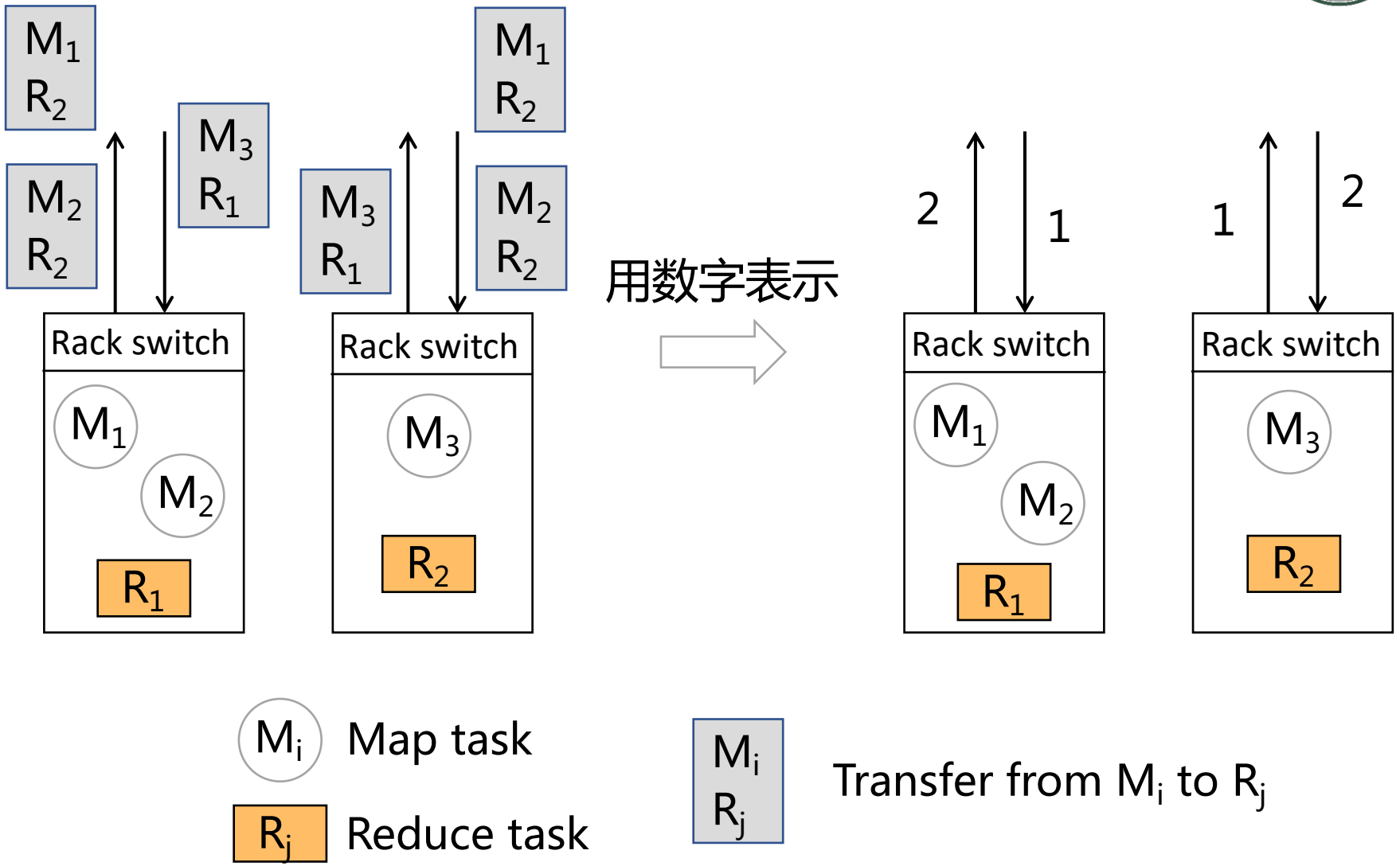
客制化的数据选择策略

- 上述情况成立的前提是选择任意一个数据块对结果是等价的
- 实际情况可添加自定义的数据选择策略

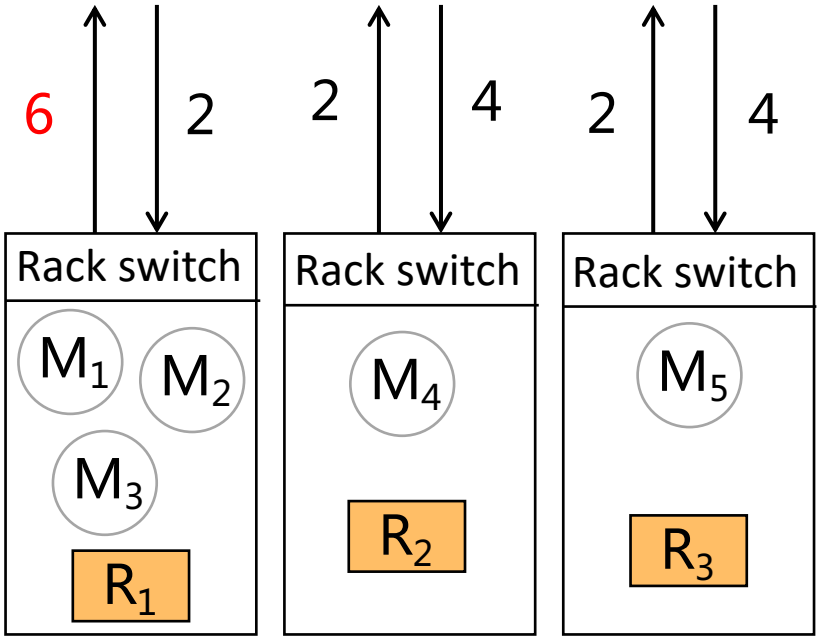
云资源管理

- 基于抽样的大数据应用
- 基于数据感知的资源调度
  - 数据局部性
  - 数据传输的均衡性

数据均衡性（1）


M1
R2
M2
R2
M3
R1
Rack switch
M1
M2
R1
M1
R2
M3
R1
M2
R2
Rack switch
M3
R2
用数字表示
2
1
Rack switch
M1
M2
R1
1
2
Rack switch
M3
R2
Mi Map task
Rj Reduce task
Mi
Rj
Transfer from Mi to Rj

数据均衡性（2）



Bottleneck link

Link with Max. transfers, i.e., 6

Cross-rack data skew

$$= \frac{Max.transfers}{Min.transfers} = \frac{6}{2} = 3$$

Data skew 越小则上游任务的结果传输越均衡，下游阶段完成的越快

Facebook种的data skew情况

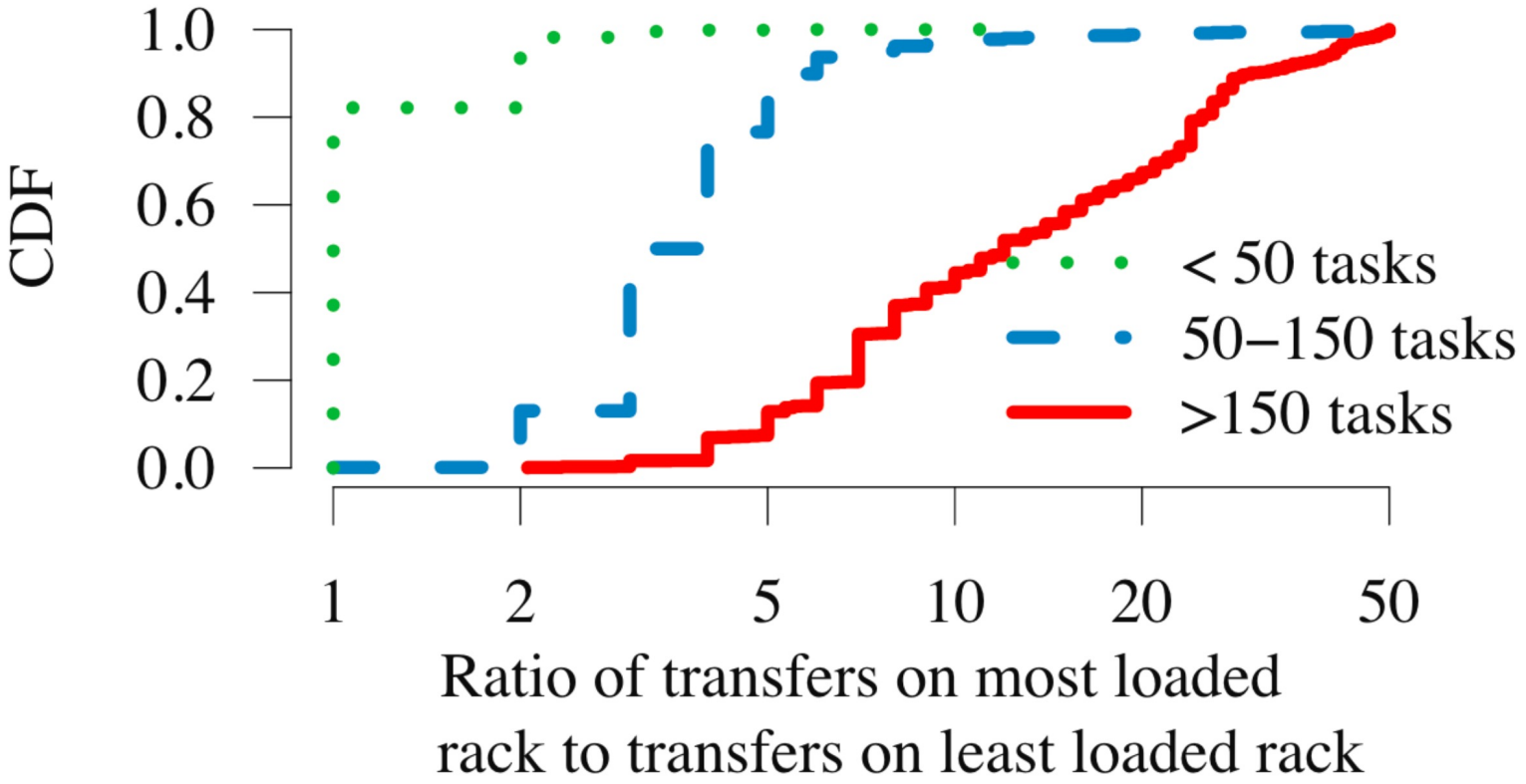


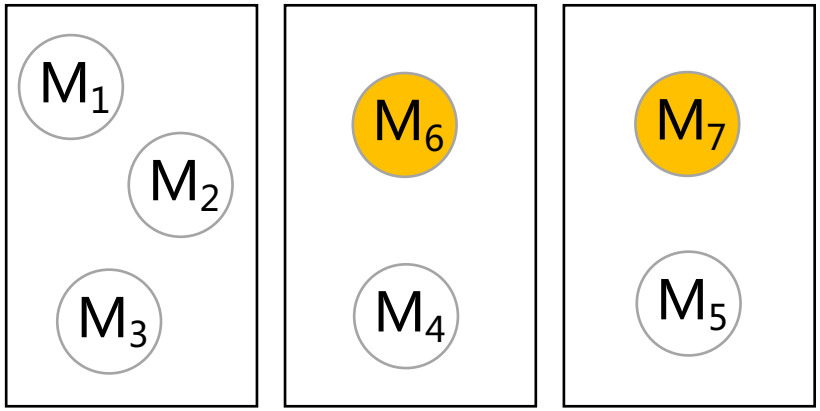
Figure 3: **CDF of cross-rack skew for the Facebook trace split by number of map tasks. Reducing cross-rack skew improves intermediate stage performance.**

额外的上游任务

- 为了缓解因 data skew 导致的数据传输瓶颈（因而使得下游任务变慢），作者提出运行额外的上游任务

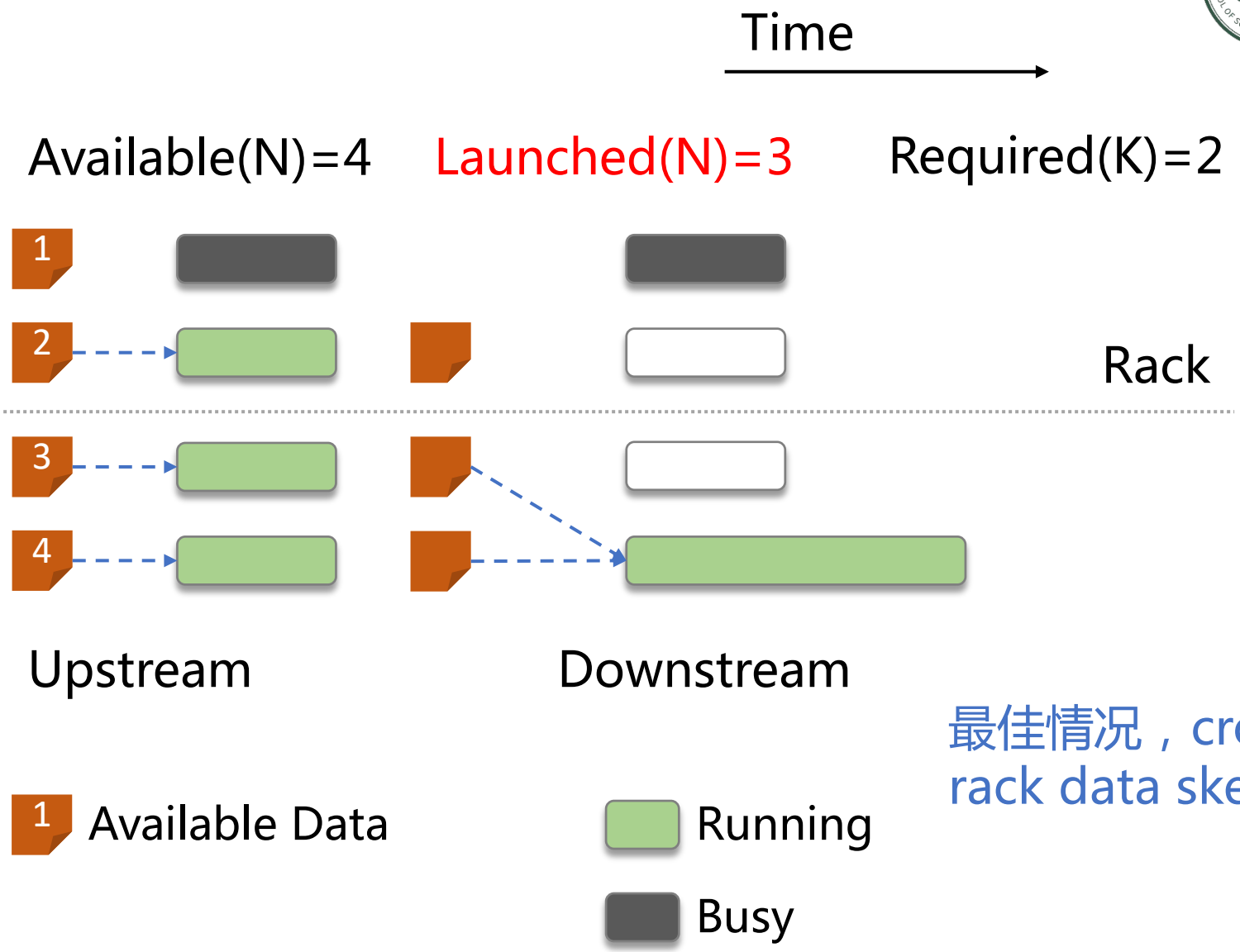
- 假设需要 K 个上游任务，算法部署 M 个任务（$M > K$）使得上游任务的输出在 rack 中的分布更加均衡

- 最终的 K 个上游任务可以在 M 个任务中选择



$K = 5, M = 7$

数据感知的选择



最佳情况，cross-rack data skew=1

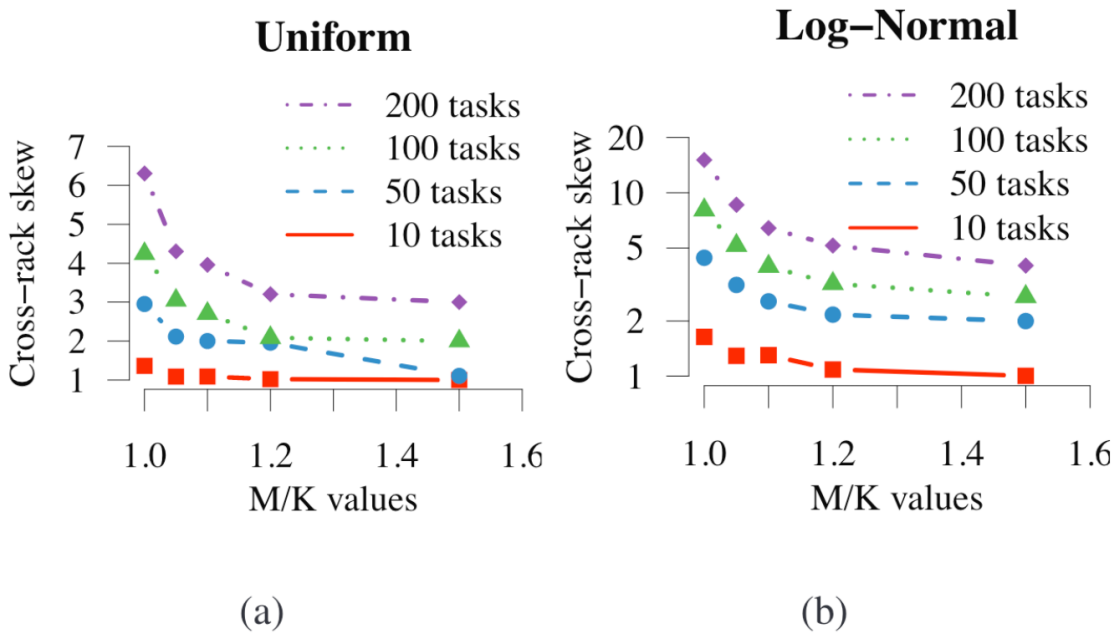
如何选择M？

M 选多大才合适呢？

作者根据合成的上游任务分布和 Facebook 的实际任务分布来研究不同 M/K 比例下的 cross-rack data skew （假定上游任务输出的大小相同，网络链路的使用率相同）

合成分布

- 将上游任务按 uniform 和 log-normal 分布部署在 100 个 rack 集群

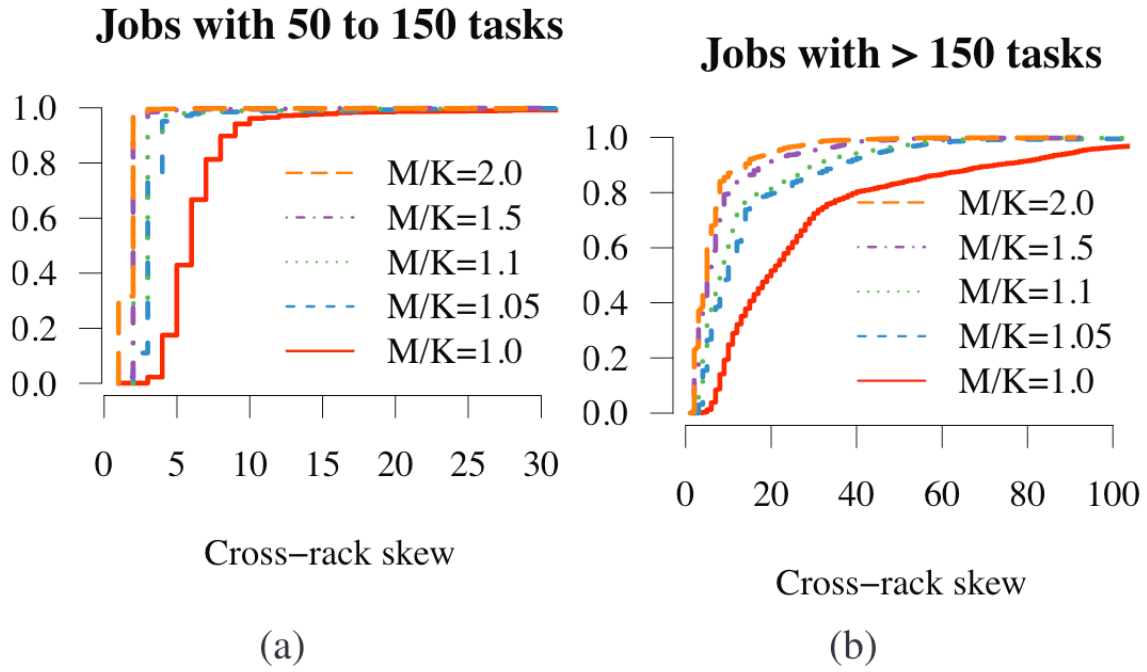


可以看到，仅需多部署 10% - 20%（$M/K = 1.1 - 1.2$）的上游任务，即可将 cross-rack skew下降超过 2 倍

Figure 6: **Cross-rack skew as we vary M/K for uniform and log-normal distributions. Even 20% extra upstream tasks greatly reduces network imbalance for later stages.**

Facebook分布

- 针对 Facebook 的实际任务分布进行 cross-rack skew 计算



当 $M/K = 2$ cross-rack skew 下降约 66%

Figure 7: **CDF of cross-rack skew as we vary M/K for the Facebook trace.**

如何选取上游任务？

- 给定 M 个上游任务，如何选择 K 个任务，使得上游任务的输出在 rank 中最均衡（即瓶颈 link 的 transfers 数量最小）？

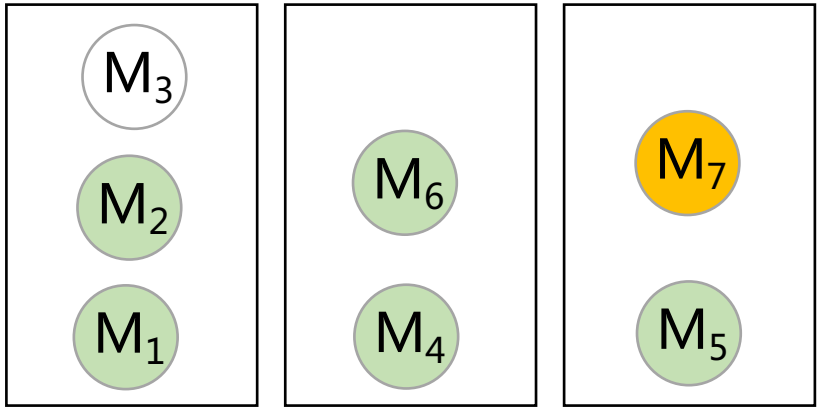
- 可证明这是一个 NP 难问题，作者提出如下启发式算法：
  - ✓构建一个哈希图，存储每个 rack 上的任务数量
  - ✓根据任务在 rack 中的索引对任务进行排序，然后根据 rack 中的任务数量进行排序
  - ✓这种排序标准确保均匀地在 rack 之间选择上游任务

任务选择算法

Algorithm 1 Choosing K upstream outputs out of M using a round-robin strategy

```
1:  Given: upstreamTasks - list with rack, index within rack for each task
2:  Given: K - number of tasks to pick
3:  // Number of upstream tasks in each rack
4:  upstreamRacksCount = map()
5:
6:  // Initialize
7:  for task in upstreamTasks do
8:      upstreamRacksCount[task.rack] += 1
9:  end for
10:
11: // Sort the tasks in round-robin fashion
12: roundRobin = upstreamTasks.sort(CompareTasks)
13: chosenK = roundRobin[0 : K]
14: return chosenK
15:
16: procedure CompareTasks(task1,task2)
17:     if task1.idx != task2.idx then
18:         // Sort first by index
19:         return task1.idx < task2.idx
20:     else
21:         // Then by number of outputs
22:         numRack1 = upstreamRacksCount[task1.rack]
23:         numRack2 = upstreamRacksCount[task2.rack]
24:         return numRack1 > numRack2
25:     end if
26: end procedure
```

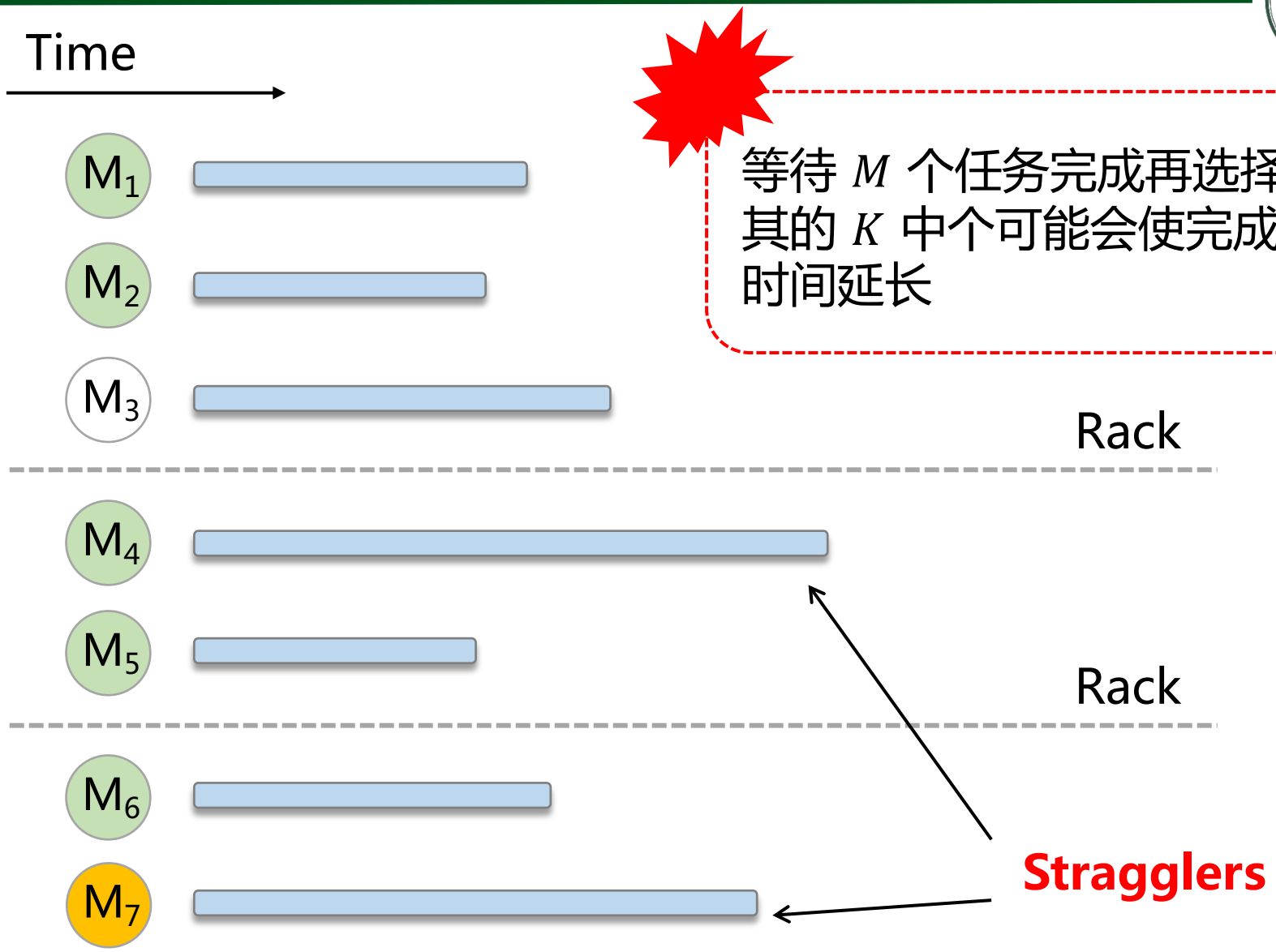
算法实例



选中的上游任务

额外部署的上游任务

落后者问题


Time
M1
M2
M3
Rack
M4
M5
Rack
M6
M7
等待 M 个任务完成再选择其的 K 中个可能会使完成时间延长
Stragglers

二者间的权衡

Stragglers **vs.** **Cross-rack data skew**

数据传输时间

- 消除 stragglers

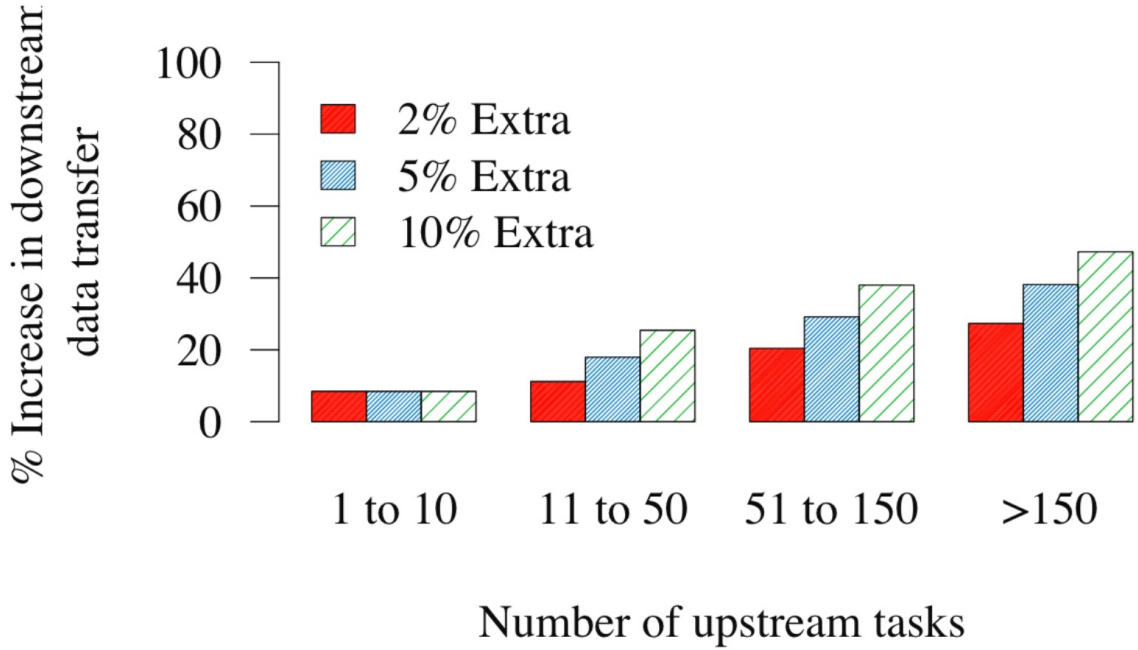


Figure 9: **Percentage of additional time spent in downstream data transfer when not using choices from 2%, 5% or 10% extra tasks. Decrease in choice increases data transfer time by** $20\% - 40\%$**.**

额外的等待时间

- 降低 cross-rack data skew

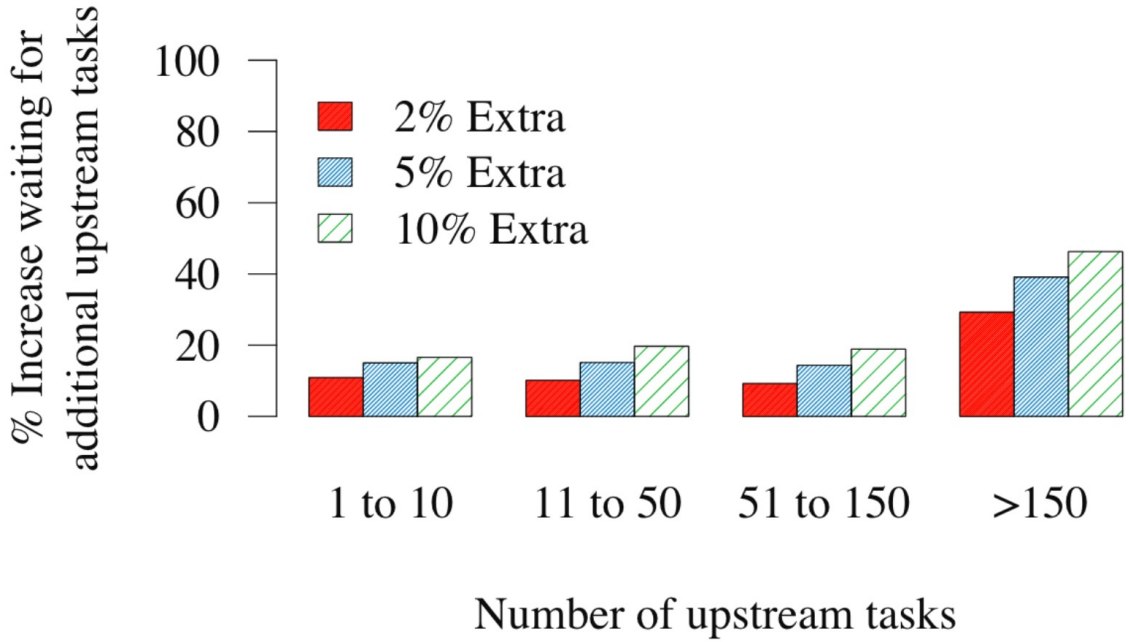


Figure 8: **Percentage of time spent waiting for additional upstream tasks when running 2%, 5% or 10% extra tasks. Stage completion time can be increased by up** $20\% - 40\%$ **due to stragglers.**

问题描述

- M 个上游任务 $u_1, u_2, \ldots, u_M$，目标是选中其中的 K 个
- 对 M 个上游任务按完成时间进行排序 $F_j \geq F_i, \forall j > i$
- 对任务 $K+1$ 到 M，记 $D_i = F_i - F_k, \forall i > k$
- 将问题转化为选择 $K'(K \leq K' \leq M)$ 个任务，使得下列时间之和最小（其中 $S_{k'}$ 为数据传输的时间）

$$T = F_k + D_{k'} + S_{k'}$$

- 分析可知，K' 越大，则 $D_{k'}$ 越大而 $S_{k'}$ 越小

启发式算法（1）

- 假设所有任务的完成时间已知，采用暴力穷举即可得到最优解

- 论文采用下列两个启发式规则避免暴力穷举
  - ✓在搜索过程开始时，记录选择最佳任务集合获得的最大改进 T_Δ，当 $D_{k'} > T_\Delta$ 时停止搜索，因为继续搜索将会增大 T
  - ✓合并完成时间足够接近（即 $D_{i+1} - D_i < \delta$ ）的任务，进一步减小搜索空间

启发式算法（2）

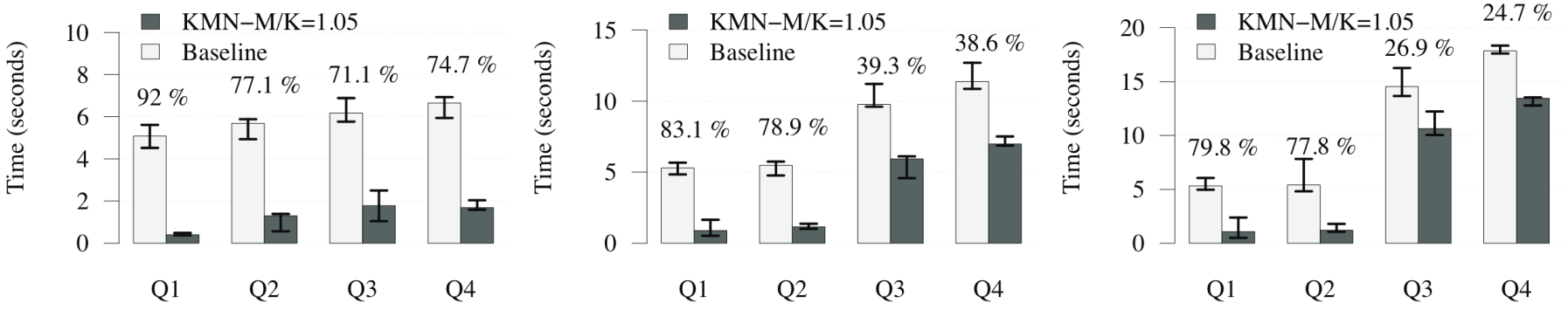
- 实际上不可能提前知道任务的完成时间，论文尝试估计方法
  - ✓根据任务的大小查找历史上是否有相同的任务，适合运行周期性任务的集群
  - ✓若历史任务无法获取，使用初始的一些任务拟合任务大小的分布，并以此估算任务的完成时间

启发式算法（3）

- 启发式算法在Facebook中的情况
  - ✓当 $M/K = 1.1$ 且 $\delta = 1\%$ 时，仅需查看 4% 的所有可能组合即可获得最优等待时间
  - ✓δ 是一个重要的变量，当 $\delta = 0.1\%$ 时，需要查看 8% 的组合
  - ✓不使用 δ 则搜索时间长达 1000ms，不可取

抽样查询

- 4 real-world sampling queries from Conviva, a video analytics company
- Sampling fraction (K/N) to be 1%, 5% and 10%



(a) Conviva queries with 1% Sampling (b) Conviva queries with 5% Sampling (c) Conviva queries with 10% Sampling

Figure 11: **Comparing baseline and KMN-1.05 with sampling-queries from Conviva. Numbers on the bars represent percentage improvement when using KMN-$M/K = 1.05$.**

随机梯度下降（1）

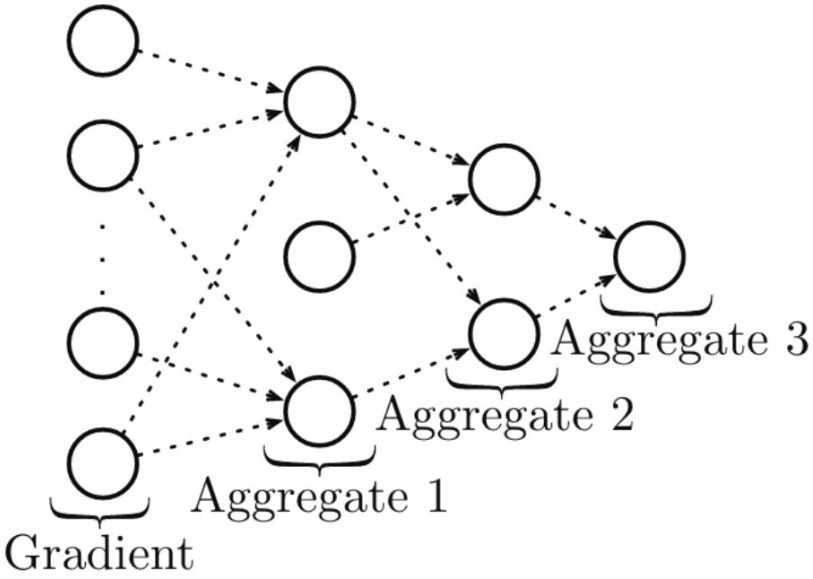


Figure 12: **Execution DAG for Stochastic Gradient Descent (SGD).**

随机梯度下降（2）

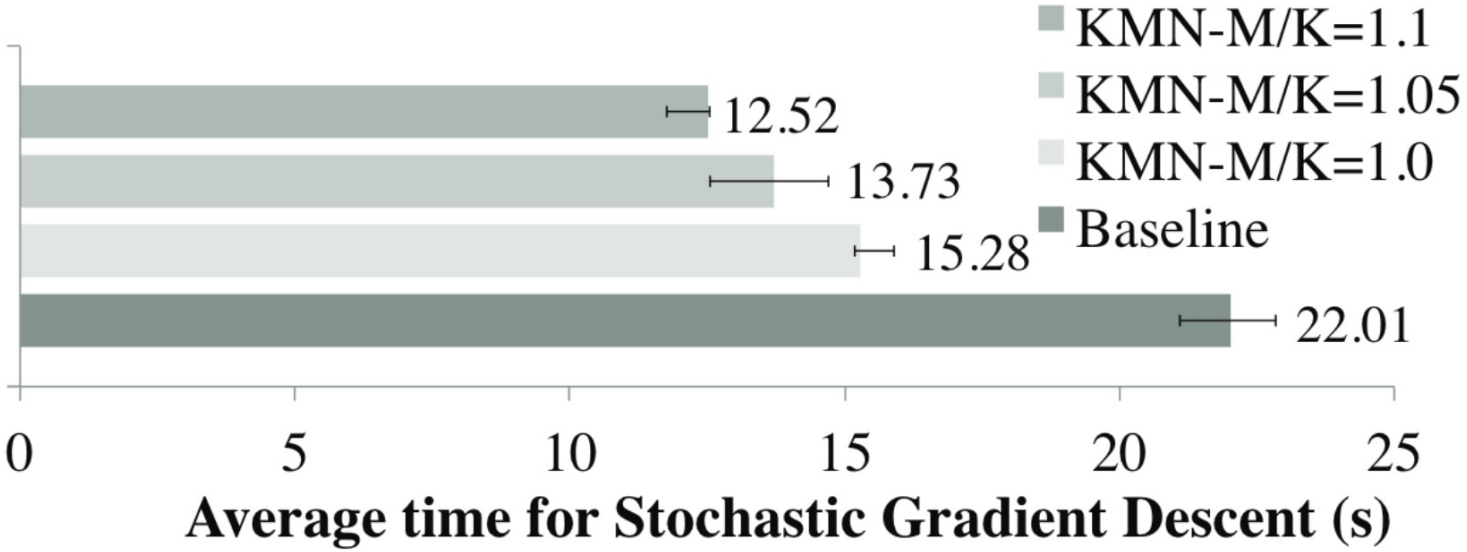


Figure 13: **Overall improvement when running Stochastic Gradient Descent using** KMN

Facebook工作负载（1）

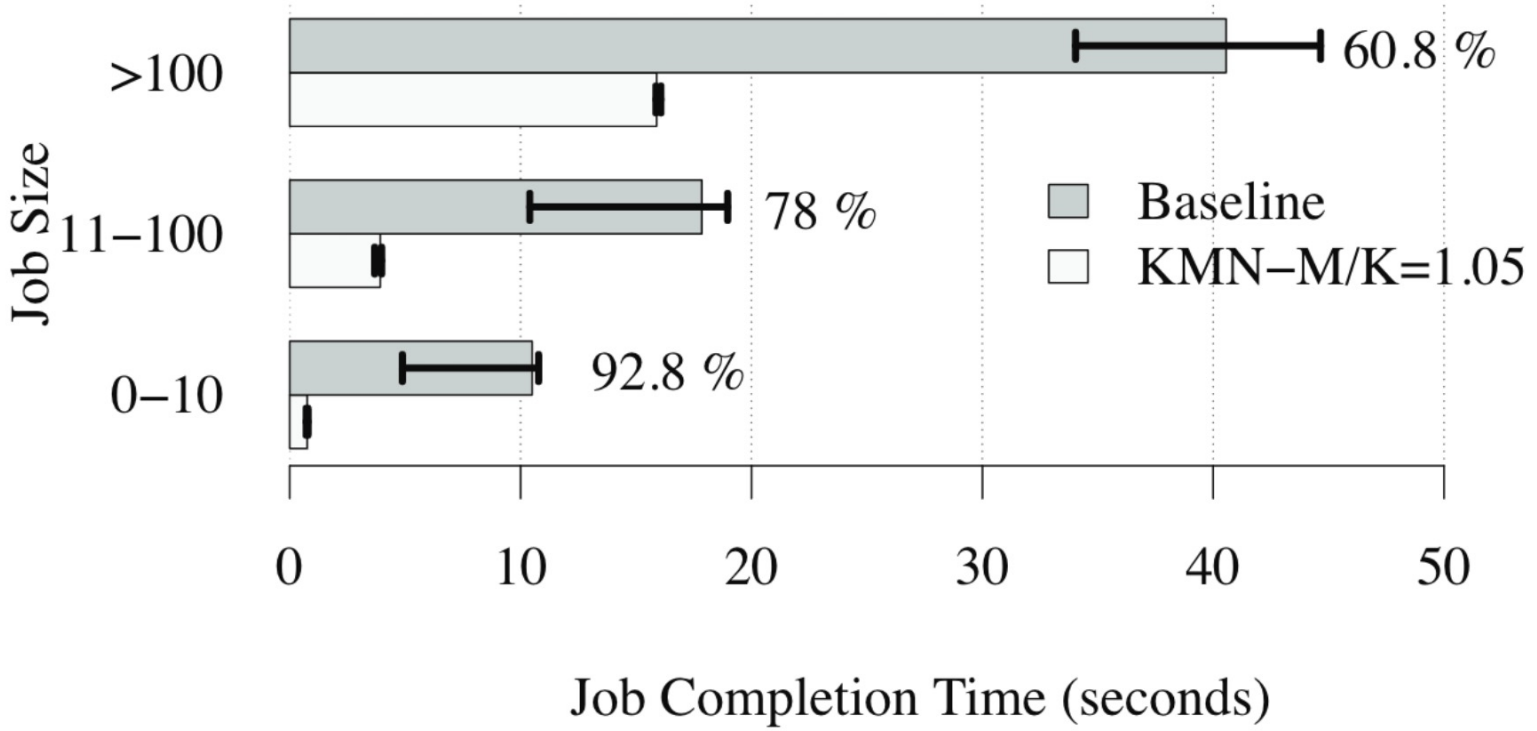


Figure 15: **Overall improvement from KMN compared to baseline. Numbers on the bar represent percentage improvement using KMN-$M/K = 1.05$.**

Facebook工作负载（2）

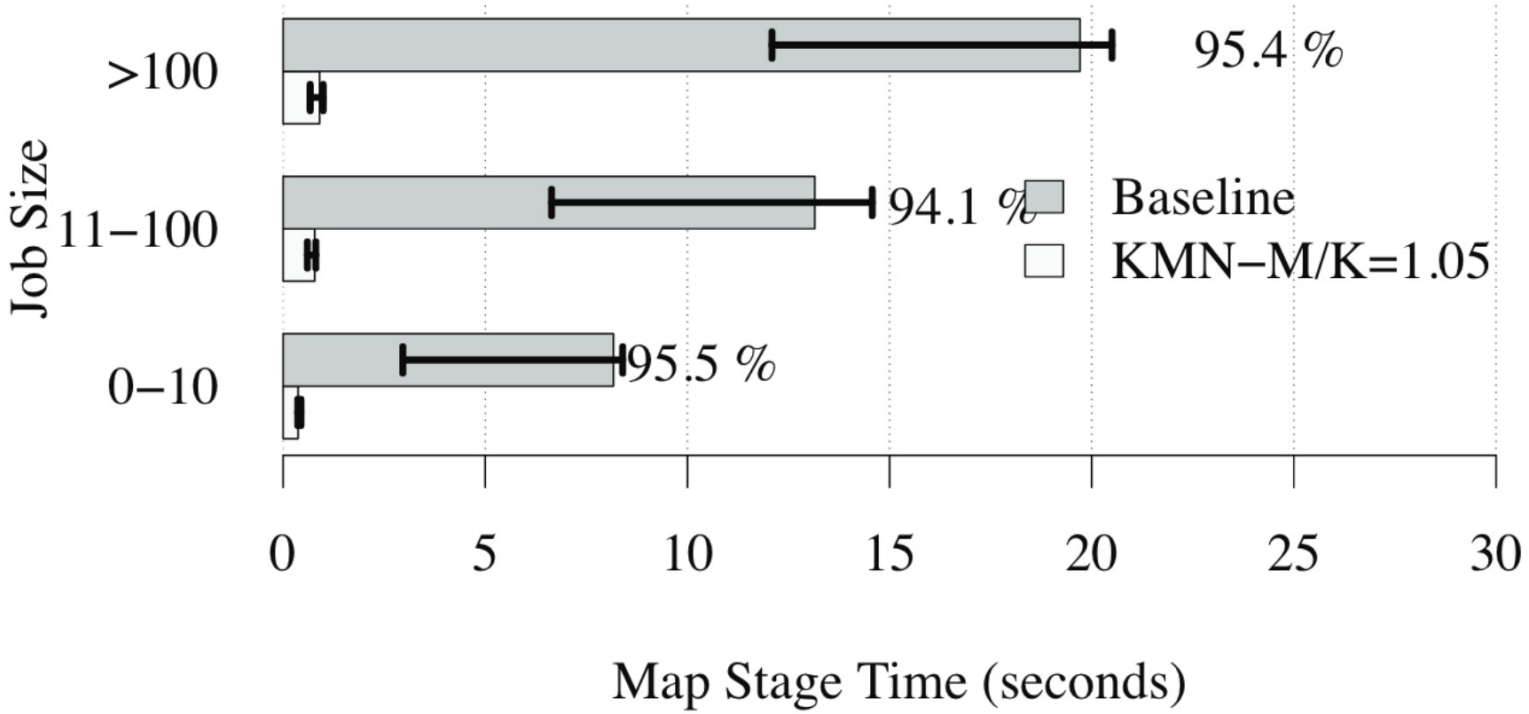


Figure 16: **Improvement due to memory locality for the Map Stage for the Facebook trace. Numbers on the bar represent percentage improvement using** KMN-$M/K = 1.05$**.**

Facebook工作负载（3）

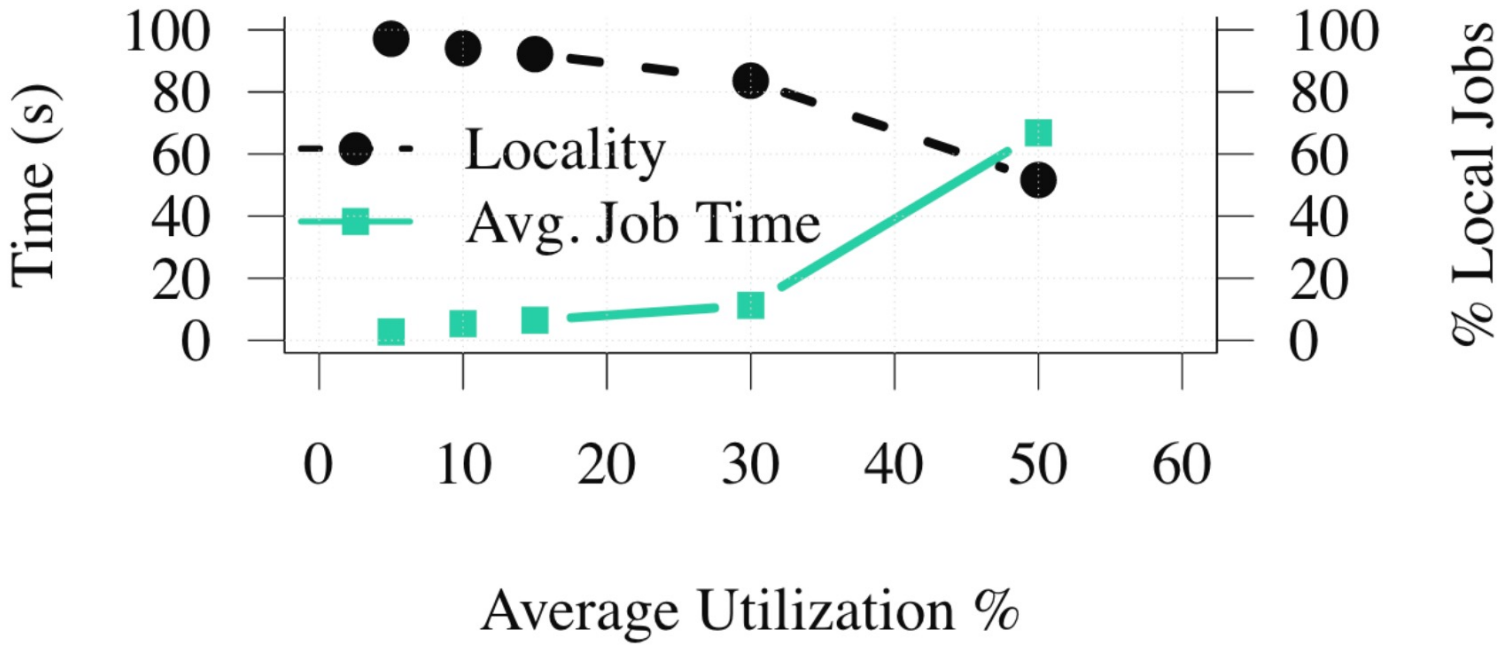


Figure 17: **Job completion time and locality as we increase utilization.**

中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬

软件工程学院

https://zbchern.github.io/sse316.html