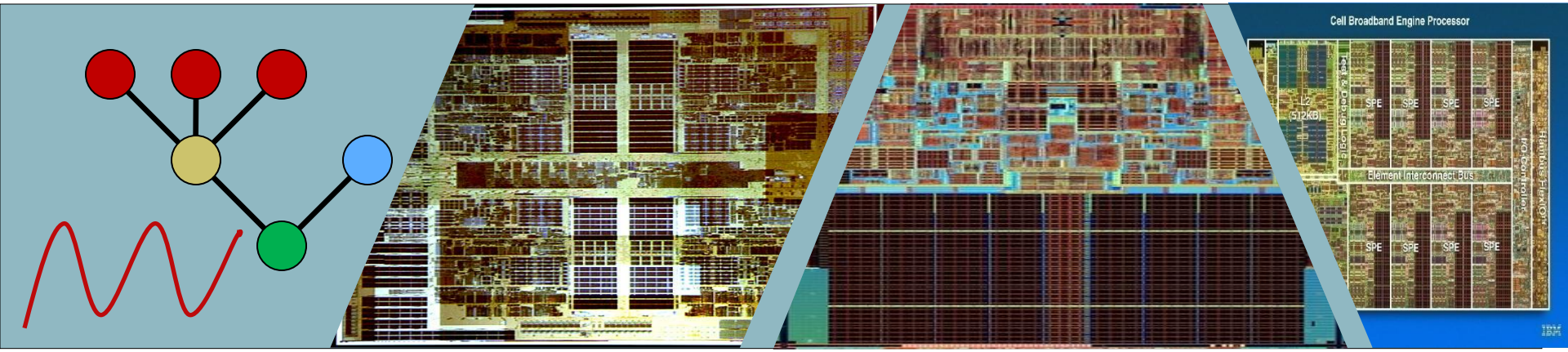
HUST

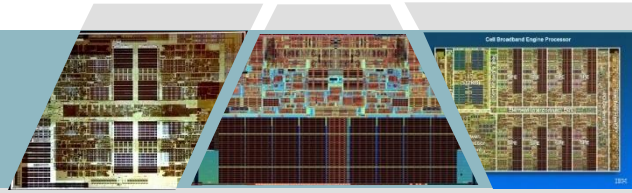
Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization



小组成员：杨 飞 M201773117

彭 昊 M201773282

内容提纲



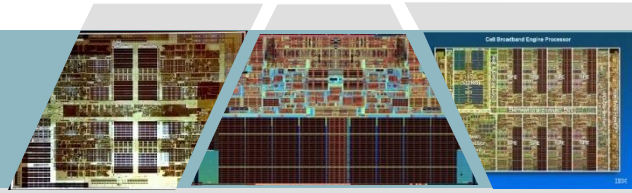
1 目标和挑战

2 研究内容

3 验证实验

4 总结与展望

目标和挑战



- 目标

  在共享集群上承载异构工作负载，以降低运营成本并实现更高的资源利用率

- 挑战
  - 提高硬件的利用率和效率
  - 异构工作负载

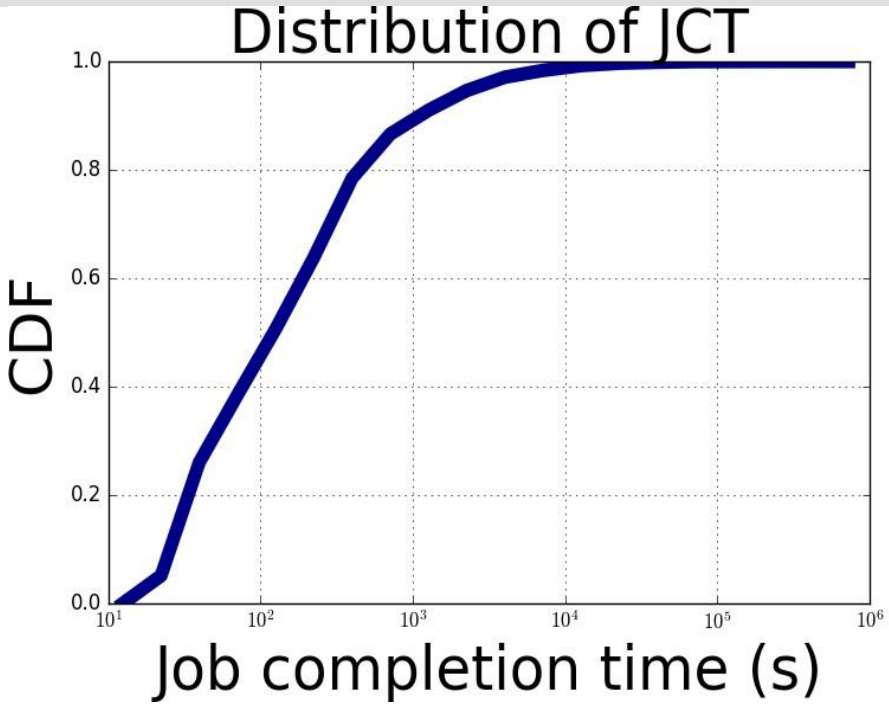
  1.资源需求不同

  ✓短作业 v.s. 长作业

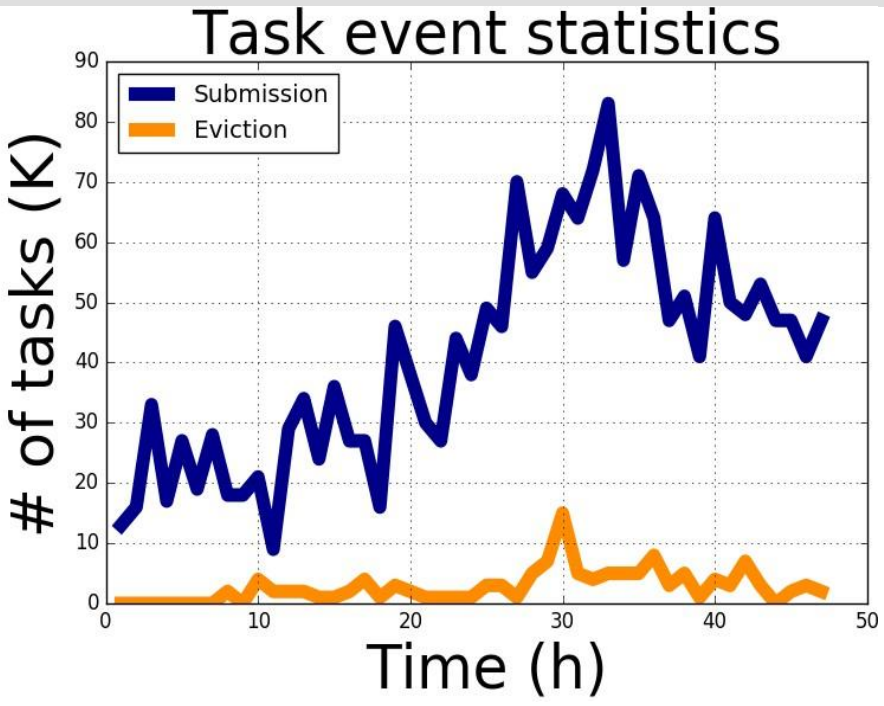
  2.服务质量需求不同

  ✓延迟 v.s. 吞吐量

研究内容



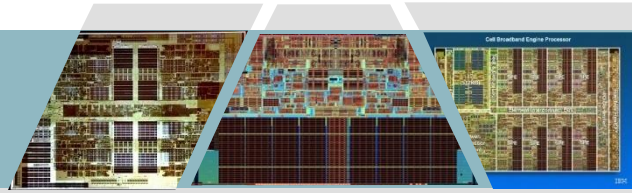
作业完成时间小于1分钟的短作业占据总共作业数量的80%左右



当集群资源达到饱和，随着任务提交的数量增多，抢占率明显上升

- 10%的长作业占据80%以上的资源

研究内容



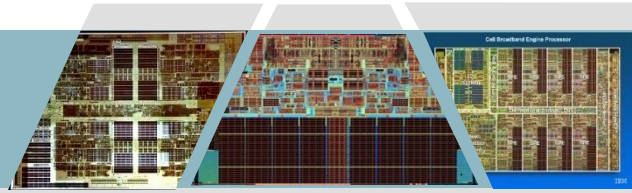
kill-based抢占方式（大部分集群调度器使用）

- 优点：简单
- 缺点：杀死的任务不能恢复，必须是重新启动

Container-based的抢占方式

- 将每个任务独立配置一个容器，使之互相隔离，然后通过cgroup来进行cpu和内存的分配
- 被抢占任务不会失去执行进度
  - ✓挂起: 从被抢占任务中回收资源
  - ✓恢复: 给该任务分配资源使得其正常工作

研究内容

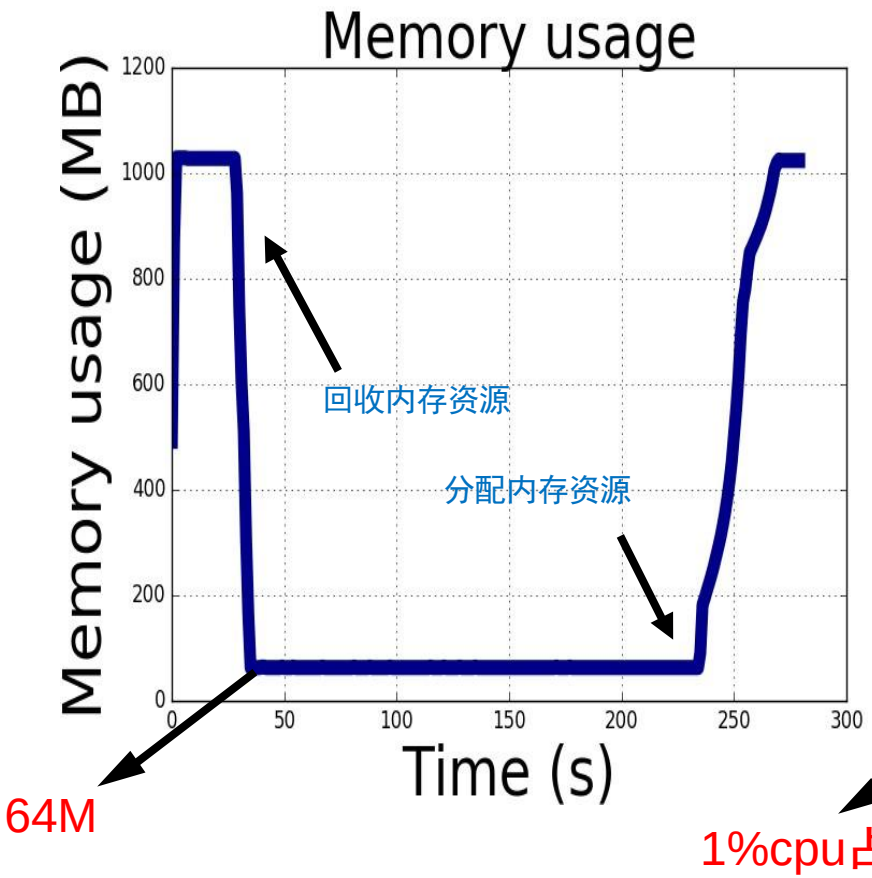


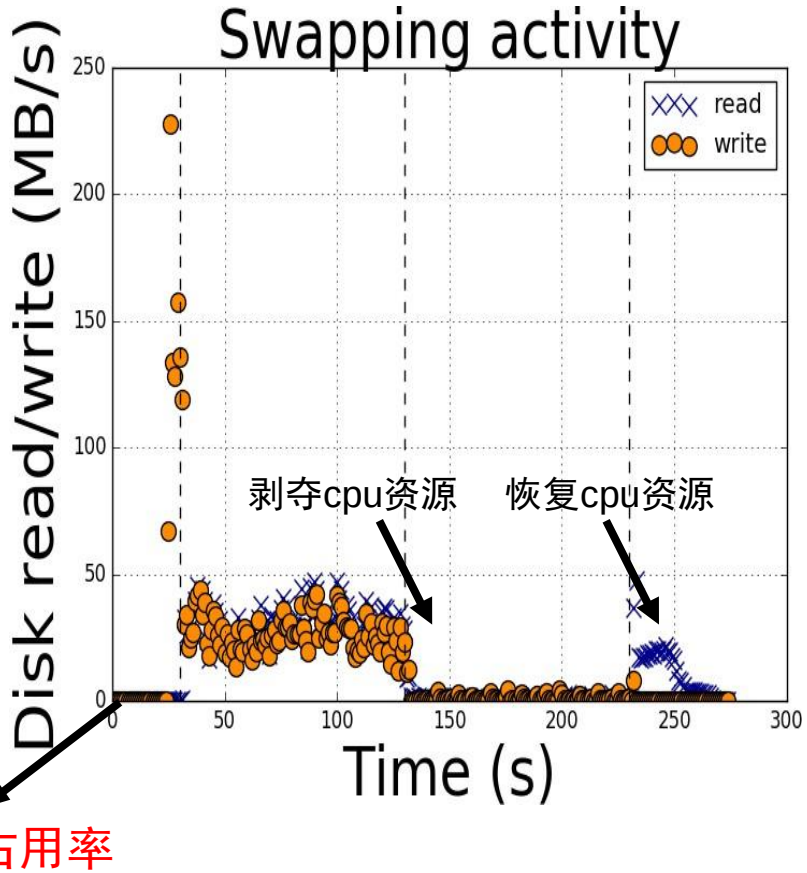
Container-based的抢占方式

- 任务容器化
  - 将任务加载到Docker容器内
  - 用cgroup控制资源分配（cpu、内存）
- 任务挂起
  - 停止任务执行: 回收cpu资源
  - 保存任务的上下文: 回收容器的内存、将所有处于内存数据写到磁盘
- 任务恢复
  - 分配任务所需的内存和cpu资源、加载上下文

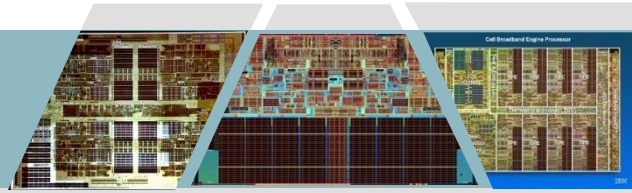
研究内容

Container-based的抢占方式





研究内容



Container-based的抢占方式

- 立即抢占

  一次性回收所有被抢占任务的资源

- 特点

  优点：简单、回收资源速度快

  缺点： 可能回收比任务所需更多的资源，导致Swapping

- 逐渐抢占

  对被抢占任务资源采用逐渐回收的方式

- 特点

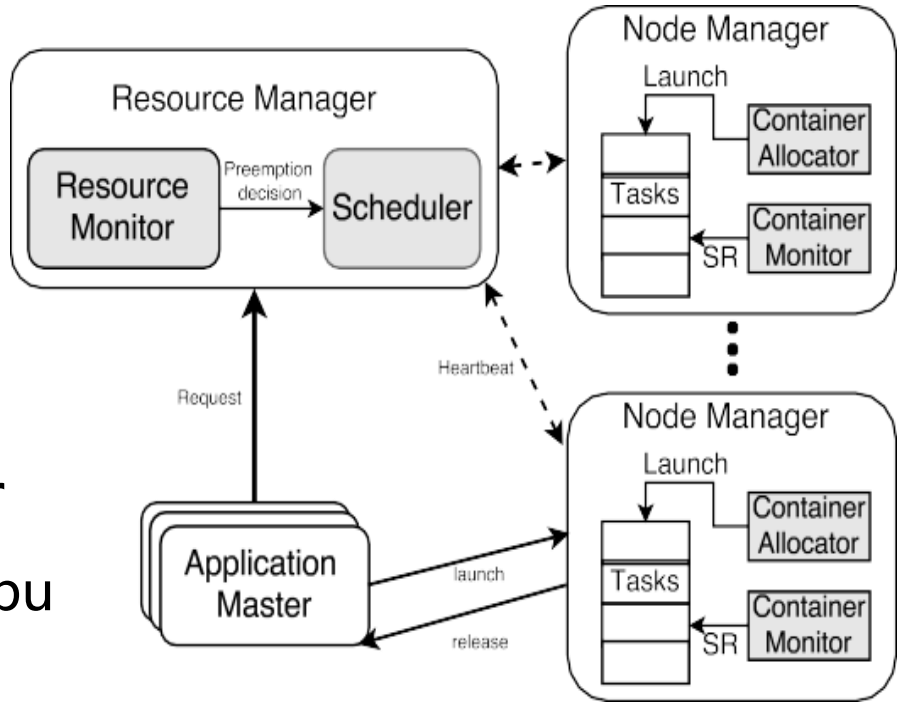
  优点：采用细粒度的资源回收机制、可以避免Swapping

  缺点： 复杂，资源回收速度慢

研究内容

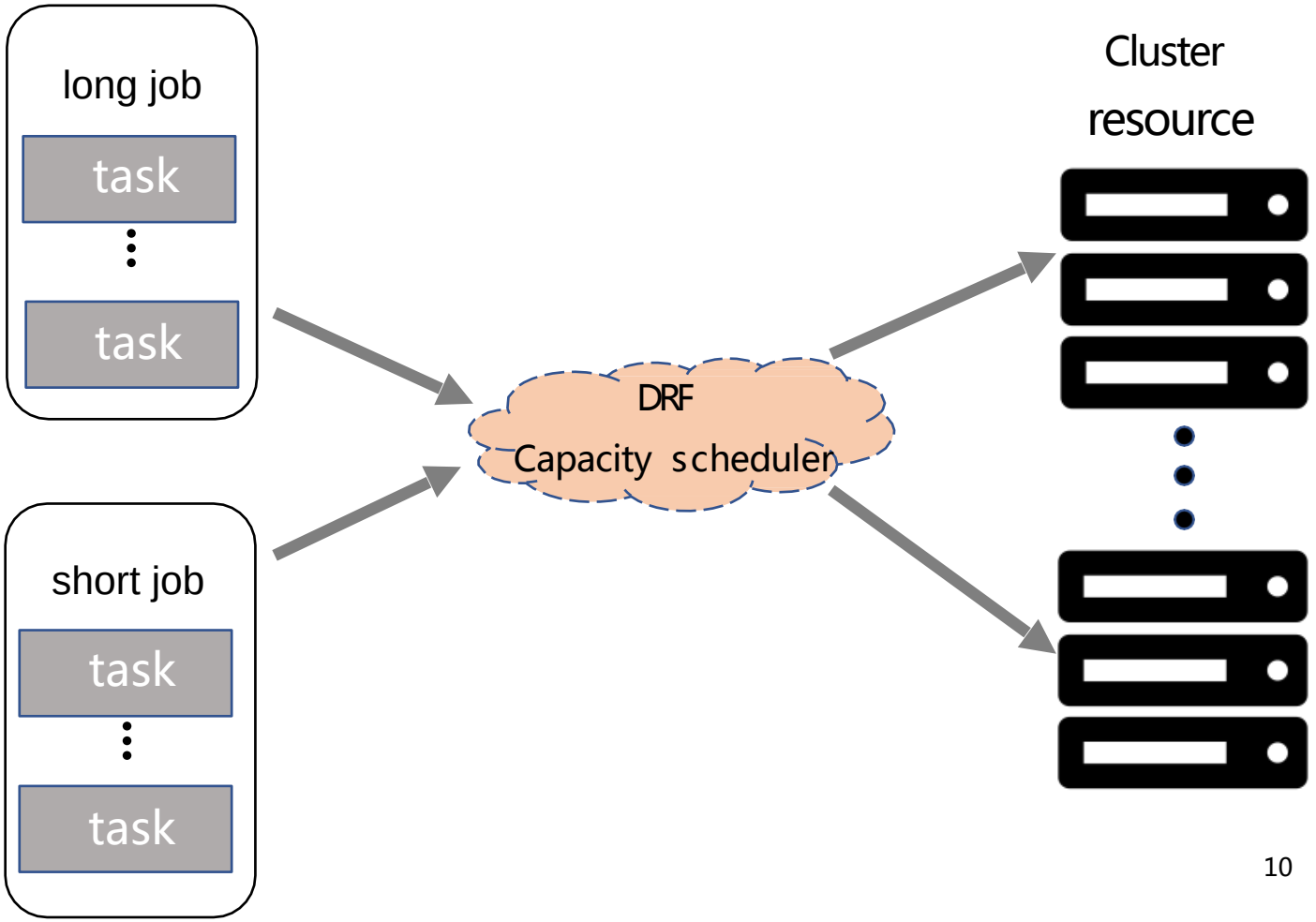
BIG-C: Preemptive Cluster Scheduling

- Container allocator
  - 用docker替换YARN的普通容器
- Container monitor
  - 执行任务挂起和恢复（S / R）操作
- Resource monitor &Scheduler
  - 决定从哪个容器抢占多少资源（cpu &内存）



研究内容

YARN's Capacity Scheduler



研究内容

Preemptive Fair Share Scheduler

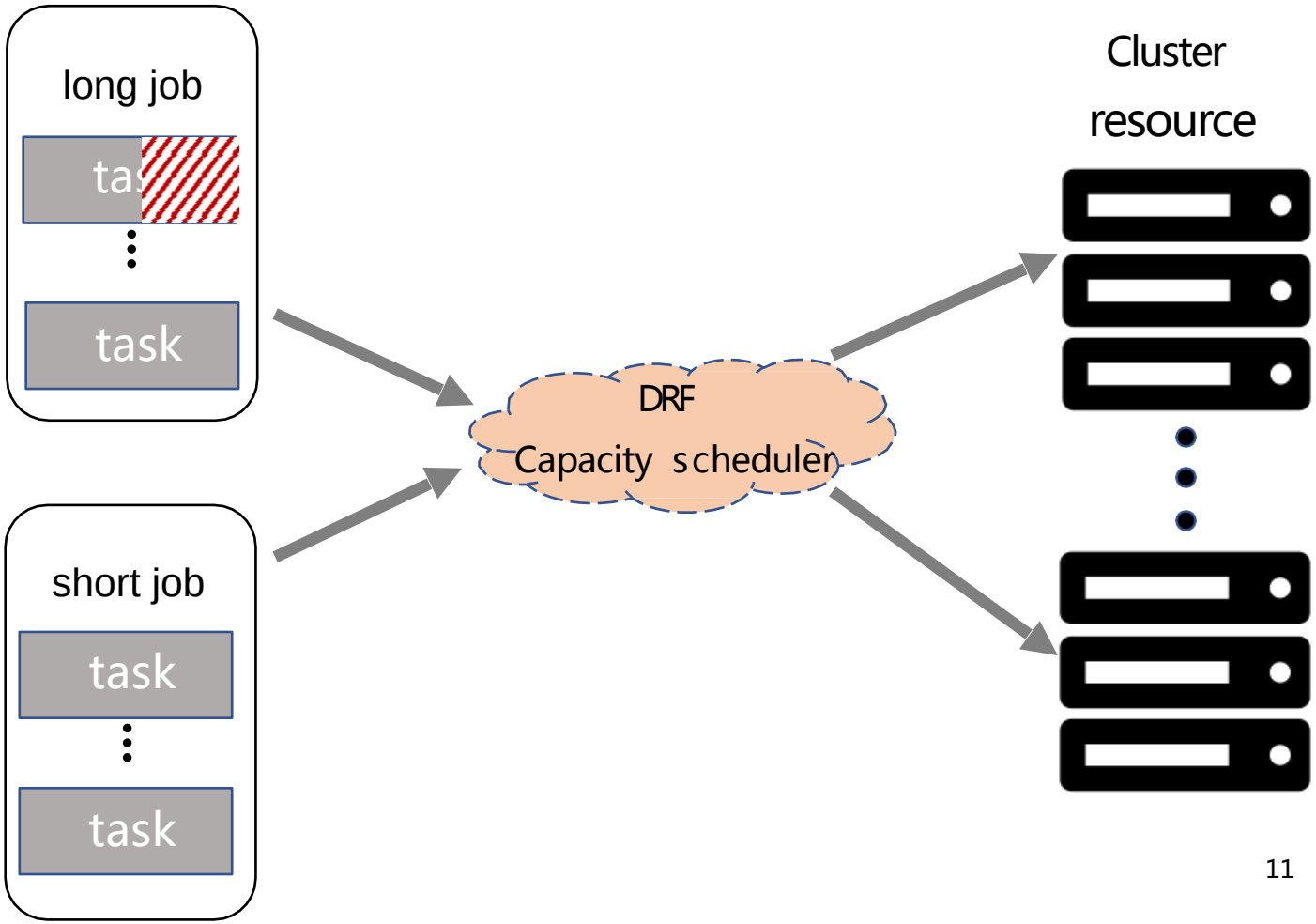
$\vec{r_l}$: long job demand
$\vec{f_l}$: long job fair share
$\vec{a}$: over-provisioned rsc
$\vec{r_s}$: short job demand
$\vec{p}$: rsc to preempt

$$\vec{a} = \vec{r_l} - \vec{f_l}$$

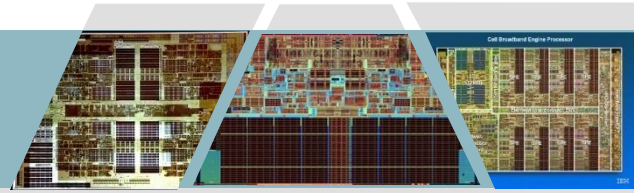
If $\vec{r_s} < \vec{a}$ • $\vec{p} = \vec{r_s}$ 抢占部分资源

else

$\vec{p}$ = **ComputeDR**($\vec{r_l}$, $\vec{a}$)



实验验证



实验环境

- 硬件

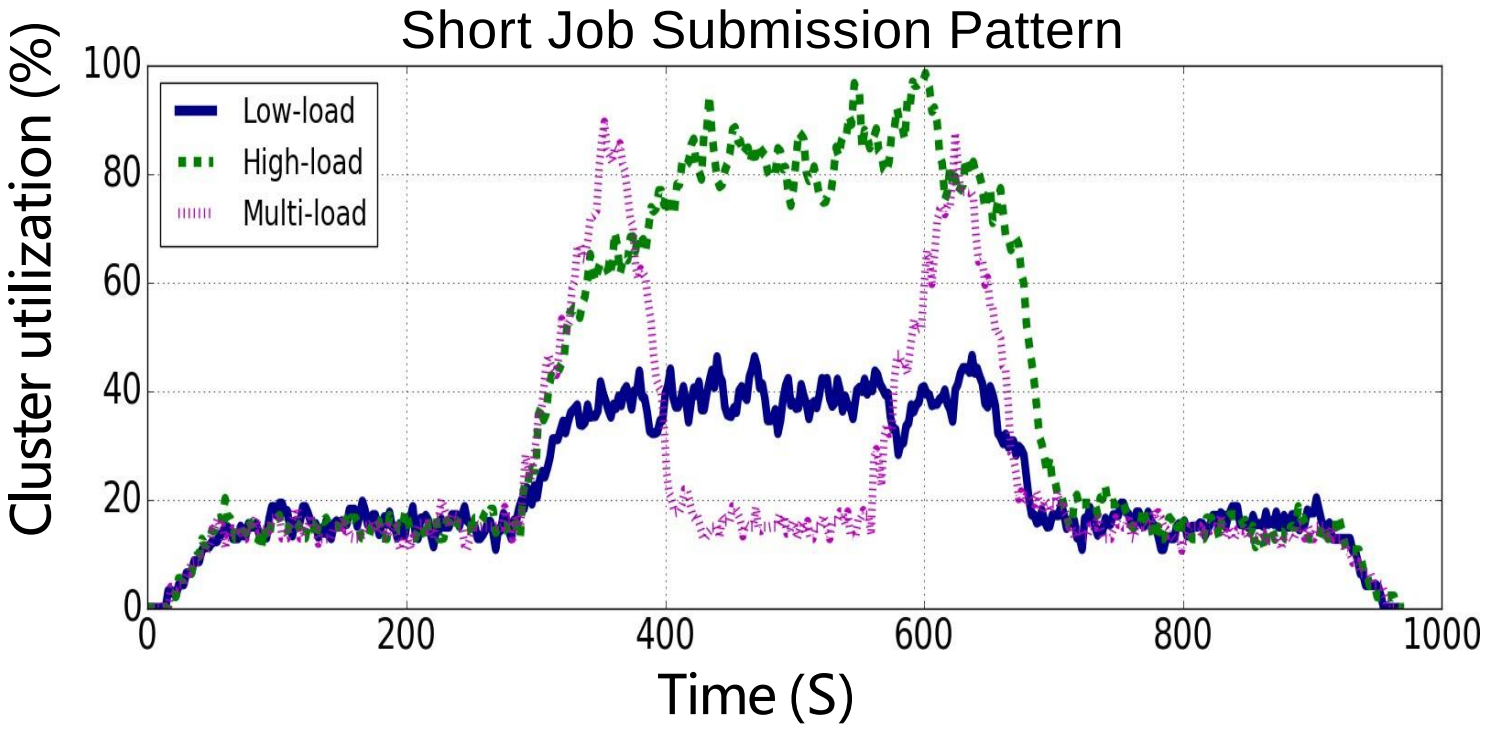
  由26个节点组成的集群、32核、128GB内存

- 软件

  采用Hadoop+Docker

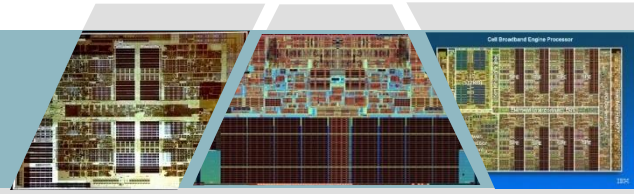
- 配置
  - 两个工作队列: 短作业占95%、长作业占 5%
  - 调度方式: 先来先服务方式（FIFO） (非抢占方式), Reserve (60% capacity for short jobs), Kill-based, 立即抢占方式 and 逐渐抢占方式

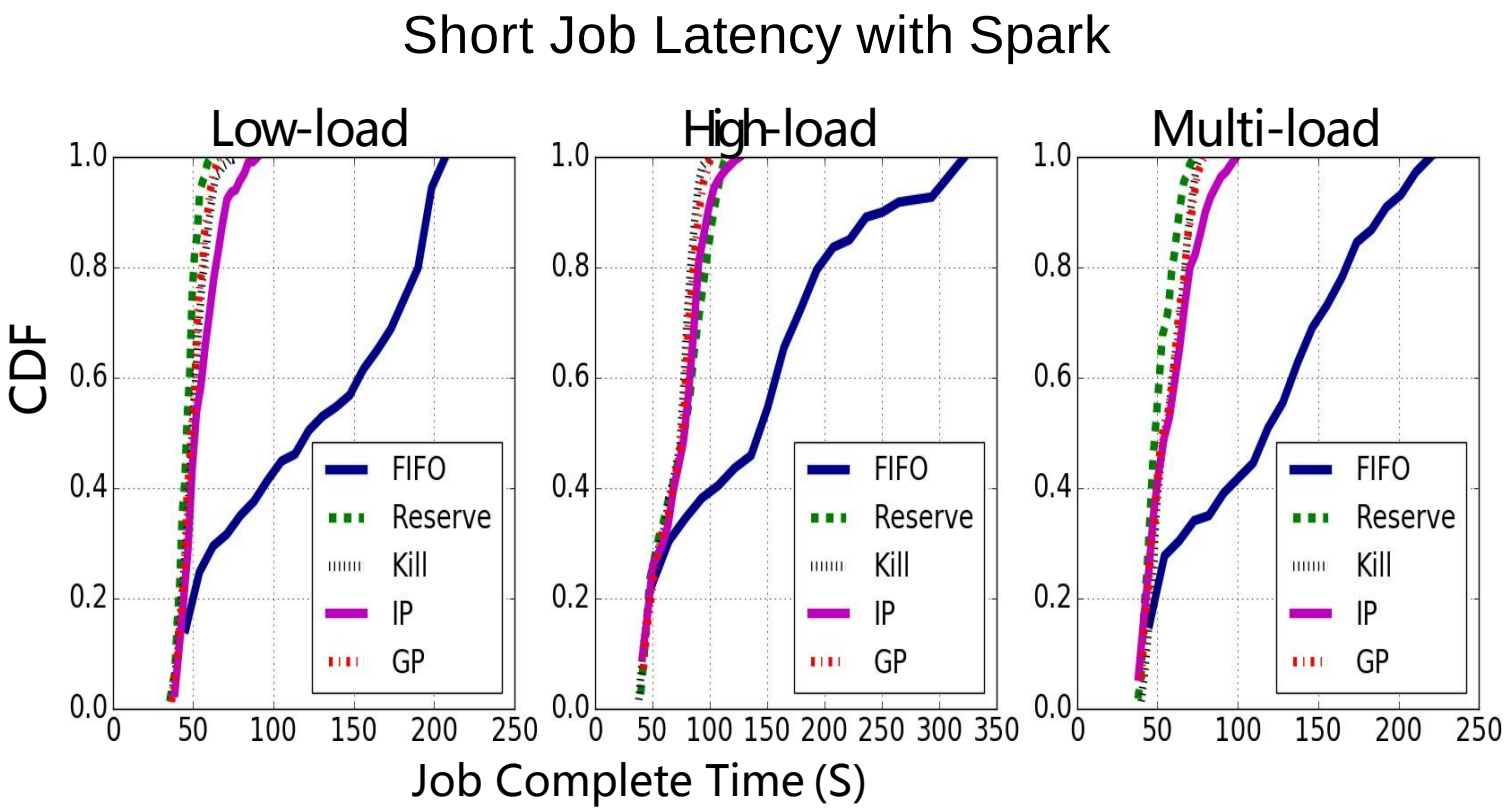
实验验证



- 对于不同负载下都包含大量的短作业，而且短作业优先级较高
- 长作业占用集群资源容量的80%左右

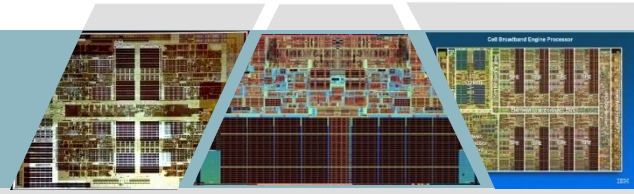
实验验证

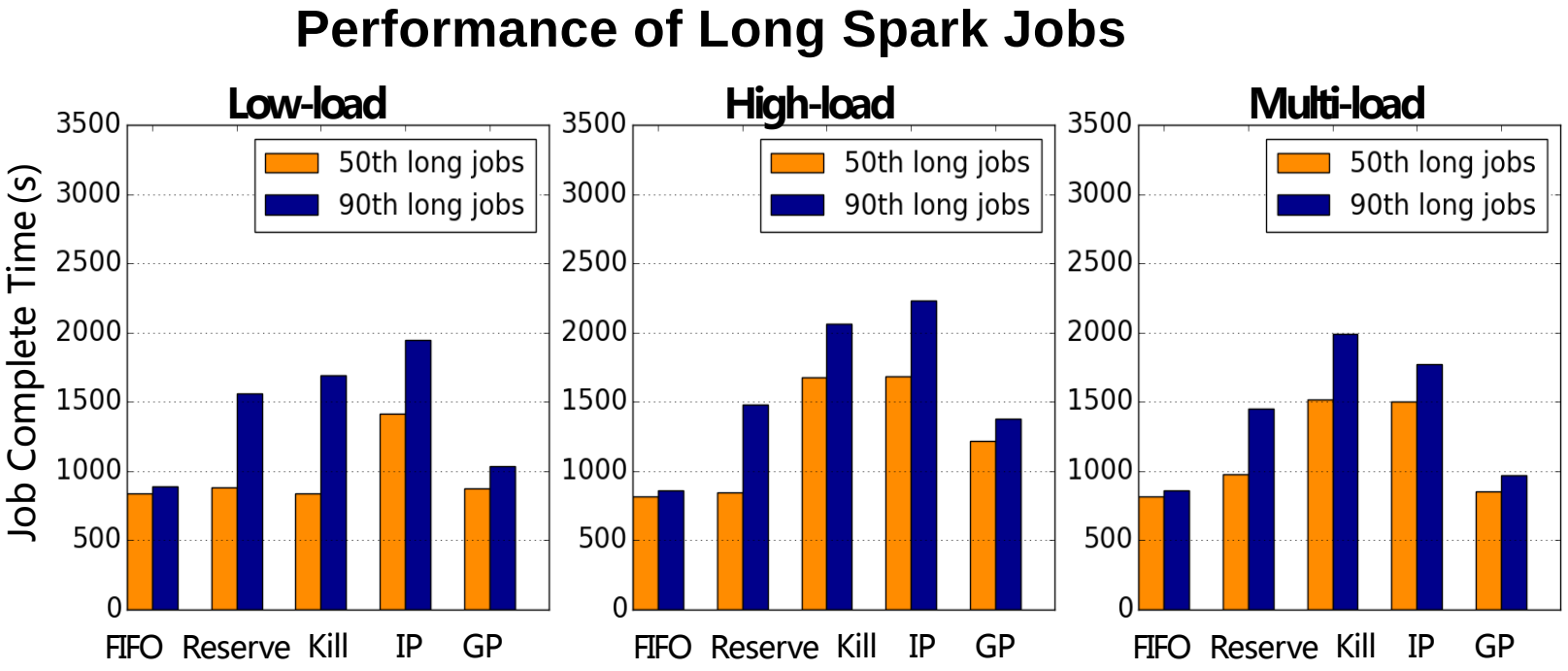




- 由于短作业不能抢占长作业的资源而处于长期等待状态导致先来先服务方式作业完成时间最长
- 由于不需要进行Swapping交换（内存与磁盘），GP（逐渐抢占方式）作业完成时间比IP（立即抢占方式）稍短

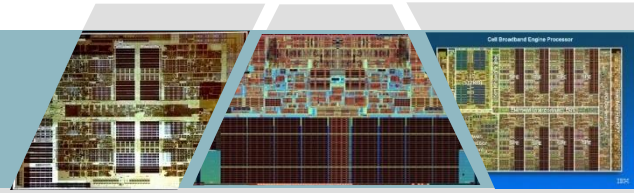
实验验证

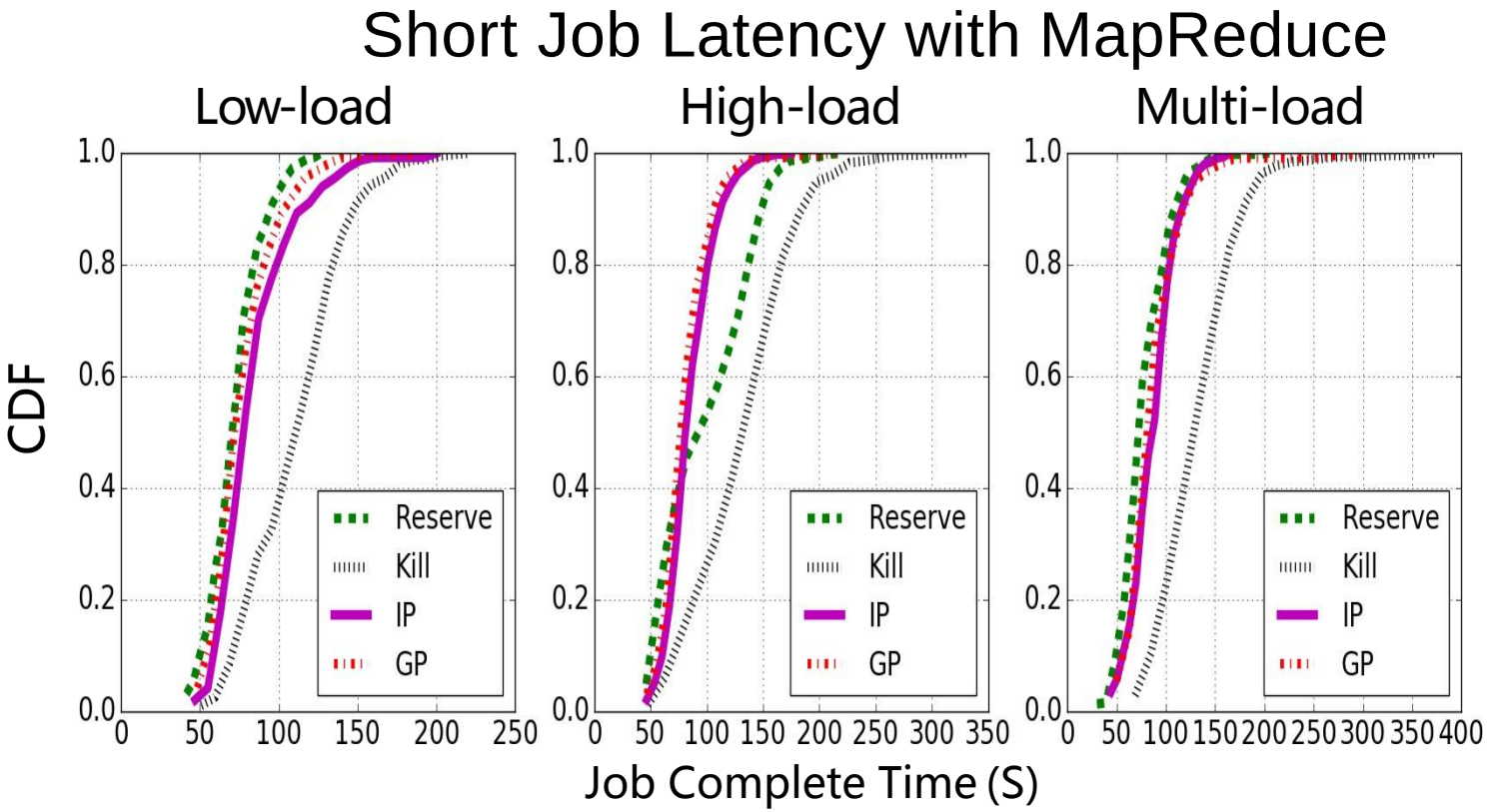




- 由于短作业不能抢占长作业资源，先来先服务方式对长作业调度表现最佳
- GP（逐渐抢占方式）的作业完成时间相对kill-based方式提高了60%左右.
- IP（立即抢占方式）在Spark Jobs下（因为需要经常的进行Swapping）反而增加了长作业的运行时间开销

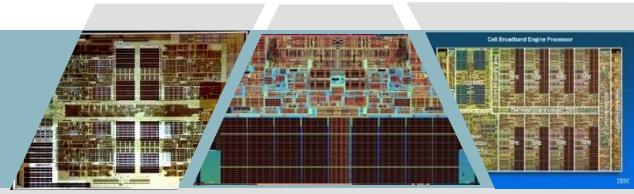
实验验证

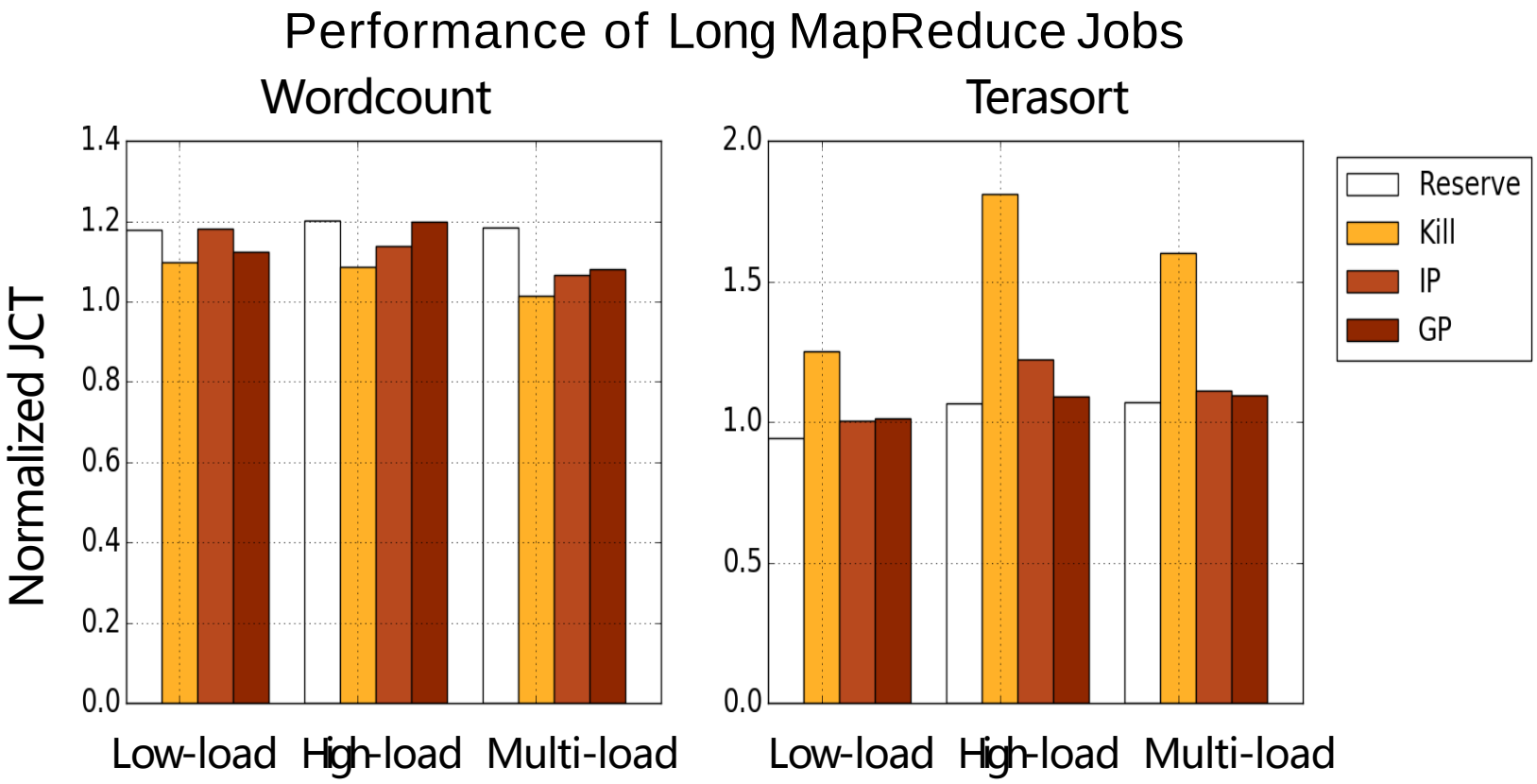




- 对于低负载和中等负载的情况下，Reserve方式表现最佳，在高负载的情况下，由于保留的资源不能满足需求从而出现性能下降
- IP和GP的短作业完成时间明显优于kill-based方式

实验验证

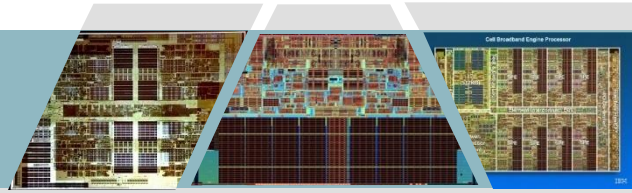




在Map程序中kill-based方式的长作业完成时间较短，在reduce中由于各个作业的高耦合，GP和IP明显由于kill-based方式

对于MapReduce程序，IP和GP方式对长作业调度性能接近

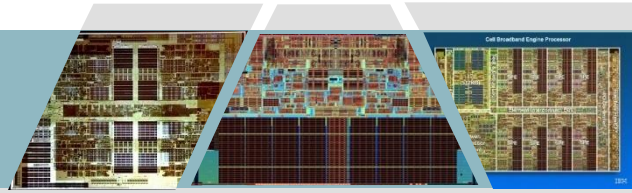
总结与展望



总结

- 集群计算缺乏有效的任务抢占机制，基于kill方式会停止长作业运行，效率较低。
- 基于Container的抢占方式是一种行之有效的集群抢占式调度
  - 轻量级，方便容器化
  - 通过精确的资源管理实现任务抢占
- 基于Container的抢占方式下，短作业调度延迟接近Reserve-based调度，长作业的完成时间接近FIFO调度，明显改善了集群的资源利用率，提升了系统的性能

总结与展望



展望

- 快速和低成本抢占是实现响应能力和高利用率的关键
- 基于容器的抢占不适合具有次秒延迟的工作负载，因为加载程序到内存中至少需要几秒时间。提供极低延迟的任务抢占是一个有趣的未来发展方向。

HUST

Thank you!