CSCI 381/780
Cloud Computing

Resource Scheduling

Jun Li
Queens College

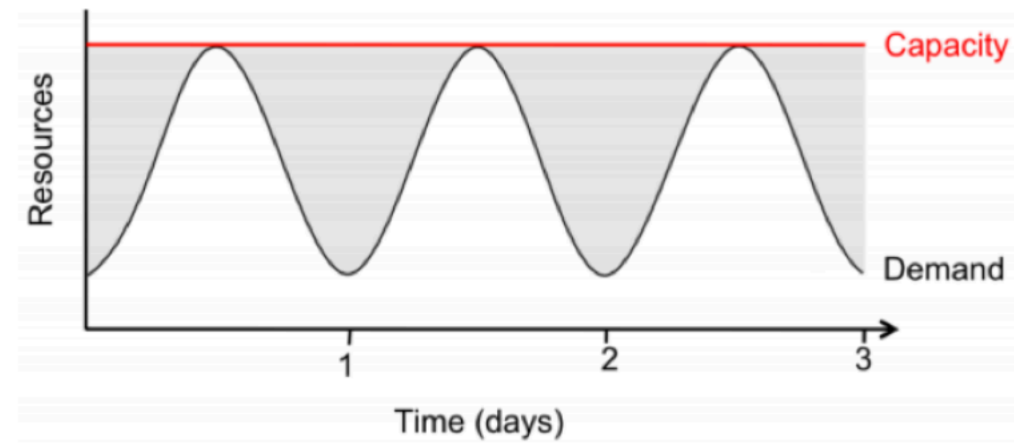
Key aspects of cloud computing

1. Illusion of infinite computing resources available on demand, eliminating need for up-front provisioning

2. The elimination of an up-front commitment

3. The ability to pay for use of computing resources on a short-term basis

Towards fuller utilization



- Source of variable demand?
  - Search, social networks, e-commerce, usage have diurnal patterns
  - Apocryphal story: AWS exists because Amazon needed to provision for holiday shopping season, wanted to monetize spare capacity
- But…if provision for peak, what around remaining time?
  - Fill-in with non-time-sensitive usage, e.g., various data crunching
  - E.g., Netflix using AWS at night for video transcoding

1. Metrics / goals for scheduling resources

2. System architecture for big-data scheduling

1. Metrics / goals for scheduling resources

2. System architecture for big-data scheduling

What do we want from a scheduler?

Isolation: have some sort of guarantee that misbehaved processes cannot affect me “too much”

Efficient resource usage: resource is not idle while there is a process whose demand is not fully satisfied

Flexibility: can express some sort of priorities, e.g., strict or time based

Single Resource: Fair Sharing

n users want to share a resource (e.g. CPU)

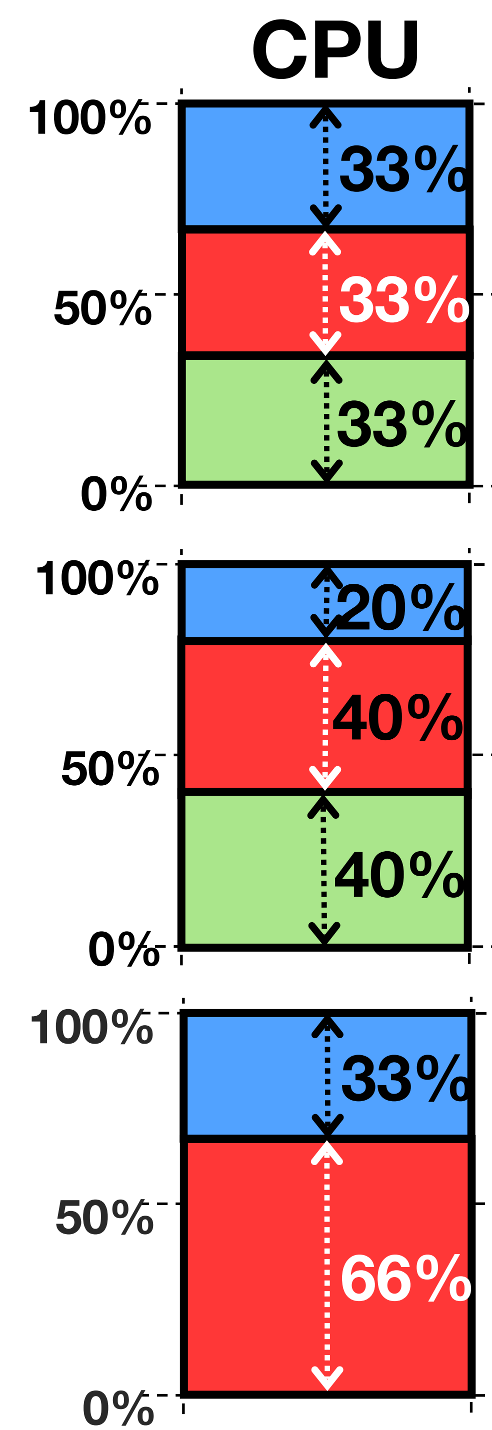
- Solution: give each *1/n* of the shared resource

Generalized by *max-min fairness*

- Handles if a user wants less than its fair share
- E.g. user 1 wants no more than 20%

Generalized by *weighted max-min fairness*

- Give weights to users according to importance
- User 1 gets weight 1, user 2 weight 2



Why Max-Min Fairness?

Weighted Fair Sharing / Proportional Shares

- User 1 gets weight 2, user 2 weight 1

Priorities

- Give user 1 weight 1000, user 2 weight 1

Reservations

- Ensure user 1 gets 10% of a resource
- Give user 1 weight 10, sum weights ≤ 100

Deadline-based scheduling

- Given a user job's demand and deadline, compute user's reservation/ weight

Isolation

- Users cannot affect others beyond their share

Widely Used

OS: proportional sharing, lottery, Linux's cfs, …

Networking: wfq, wf2q, sfq, drr, csfq, ...

Datacenters: Hadoop's fair sched, capacity sched, Quincy

Fair Queueing: Max-min Fairness implementation originated in

Fair queueing explained in a fluid flow system: reduces to bit-by-bit round robin among flows

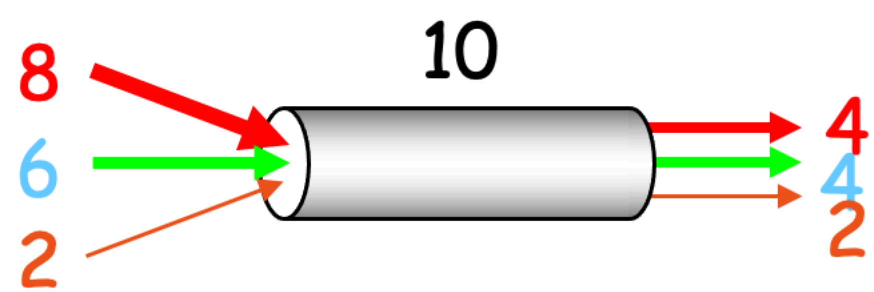
- Each flow receives $min(r_i, f)$, where
  - r_i – flow arrival rate
  - f – link fair rate (see next slide)

Weighted Fair Queueing (WFQ) – associate a weight with each flow [Demers, Keshav & Shenker '89]

Fair Rate Computation

If link congested, compute f such that

$$\sum_i \min(r_i, f) = C$$



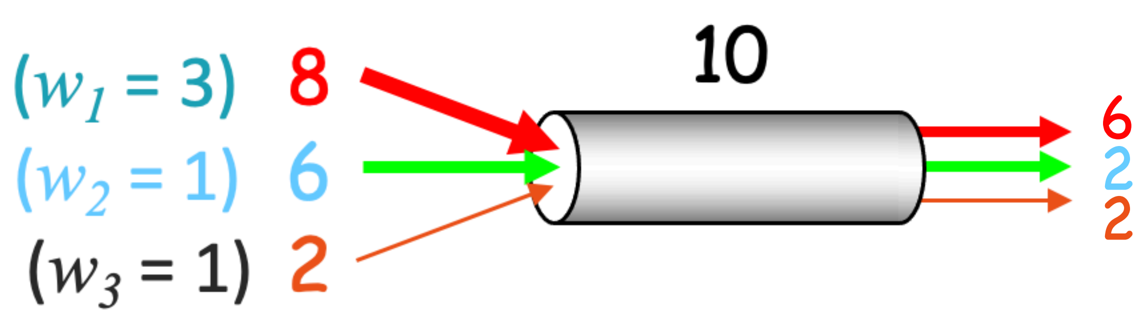
$f = 4$:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Fair Rate Computation

Associate a weight w_i with each flow i

If link congested, compute f such that

$$\sum_i \min(r_i, f \times w_i) = C$$



$f = 2$:
min(8, 2*3) = 6
min(6, 2*1) = 2
min(2, 2*1) = 2

Fluid Flow System

Flows can be served one bit at a time

- Fluid flow system, also known as Generalized Processor Sharing (GPS) [Parekh and Gallager ‘93]

WFQ can be implemented using bit-by-bit weighted round robin in GPS model

- During each round from each flow that has data to send, send a number of bits equal to the flow’s weight

Thoerethical Properties of Max-Min Fairness

Share guarantee

- Each user gets at least 1/n of the resource
- But will get less if her demand is less

Strategy-proof

- Users are not better off by asking for more than they need
- Users have no reason to lie

Cheating the Scheduler

Users willing to *game* the system to get more resources

Real-life examples

- A cloud provider had quotas on map and reduce slots
  Some users found out that the map-quota was low.
  Users implemented maps in the reduce slots!

- A search company provided dedicated machines to users that could ensure certain level of utilization (e.g. 80%).
  Users used busy-loops to inflate utilization

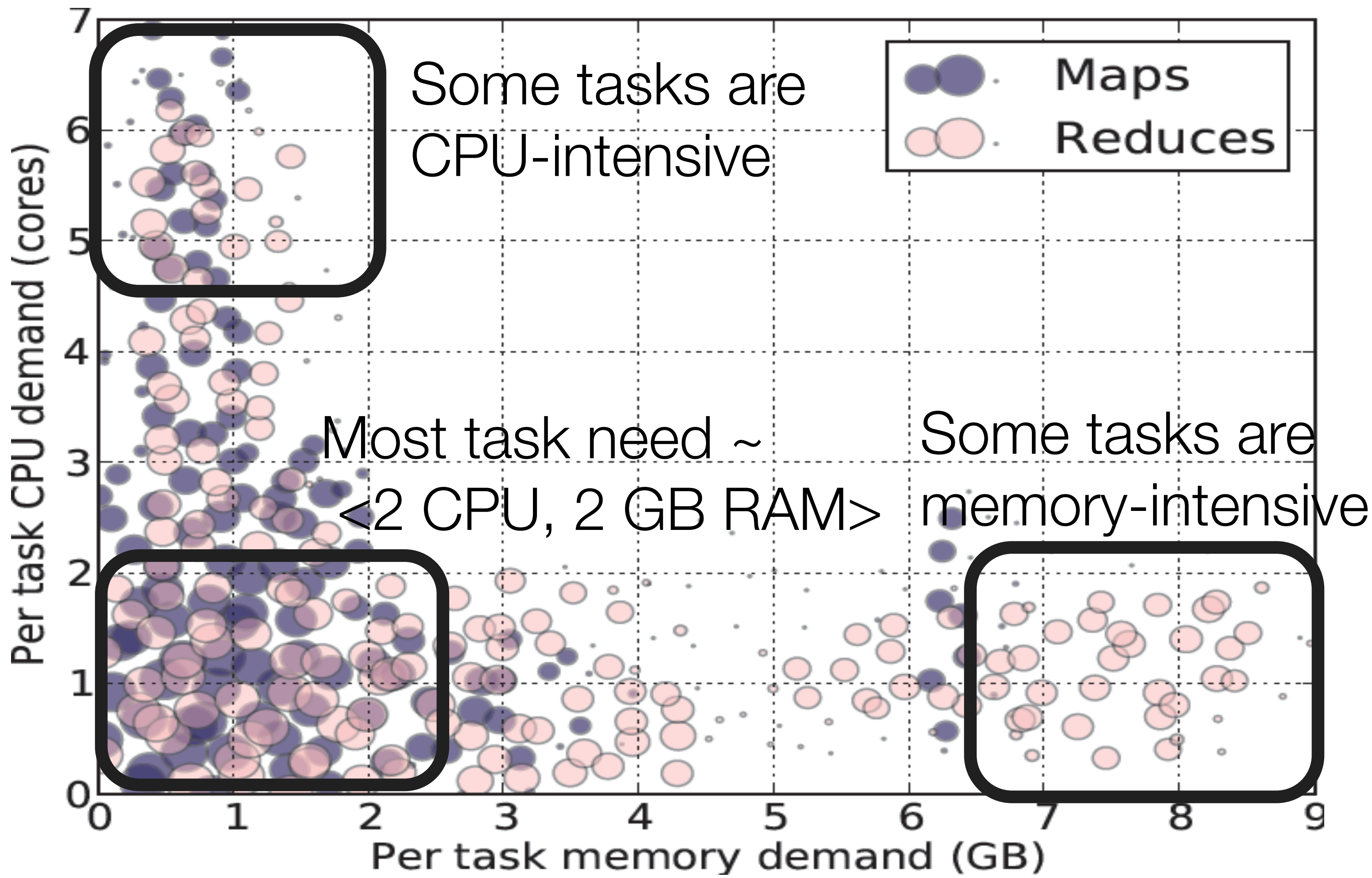
Why is Max-Min Fairness Not Enough?

Job scheduling is not only about a *single* resource

- Tasks consume CPU, memory, network and disk I/O

What are task demands today?

Heterogeneous Resource Demands



2000-node Hadoop Cluster at Facebook (Oct 2010)

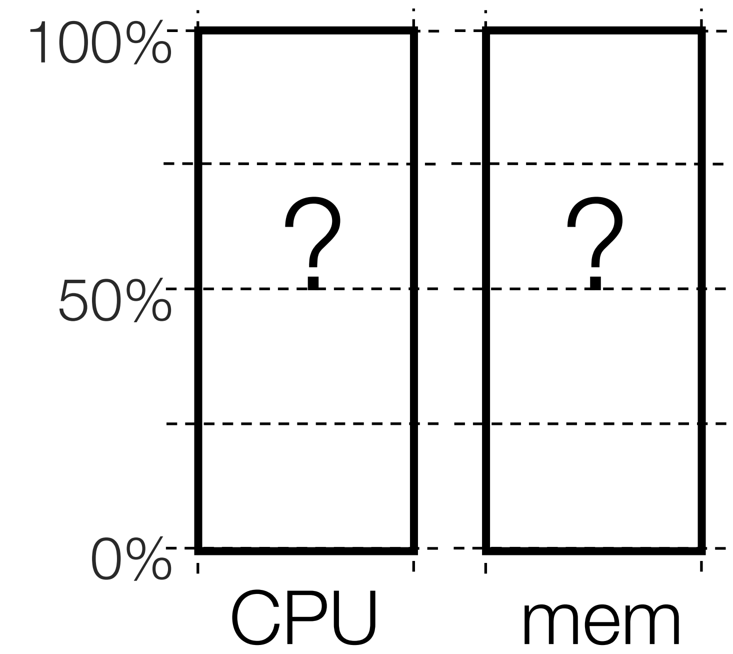
Problem

2 resources: CPUs & mem

User 1 wants <1 CPU, 4 GB> per task

User 2 wants <3 CPU, 1 GB> per task

What's a fair allocation?

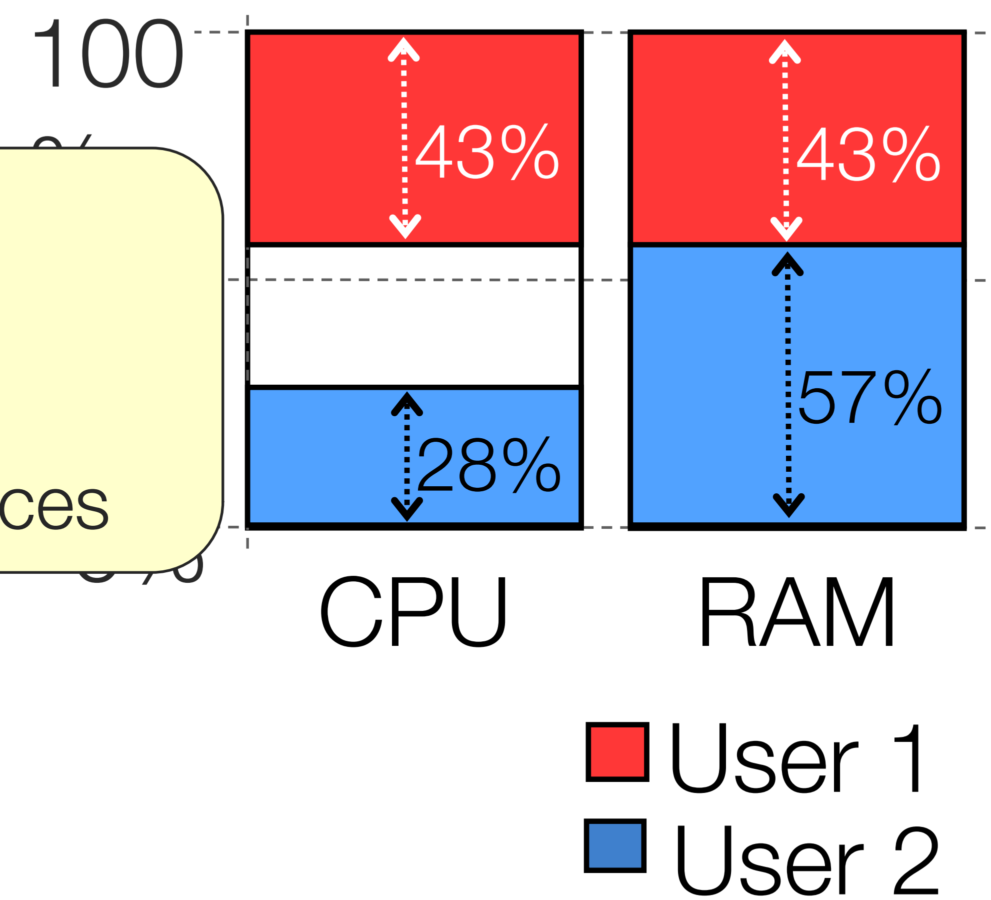


A Natural Policy

Asset Fairness

- Equalize each user's *sum of resource shares*

Cluster with 28 CPUs, 56 GB RAM



Problem: violates share guarantee
User 1 has < 50% of both CPUs and RAM

Better off in a separate cluster with half the resources

Asset fairness yields

- U_1: 12 tasks: <43% CPUs, 43% RAM> (∑=86%)
- U_2: 8 tasks: <28% CPUs, 57% RAM> (∑=86%)

Challenge

Can we find a fair sharing policy that provides

- Share guarantee
- Strategy-proofness

Can we generalize max-min fairness to multiple resources?

Dominant Resource Fairness (DRF)

A user's *dominant resource* is the resource user has the biggest share of

- Example:

Total resources:	8 CPU	5 GB
User 1's allocation:	2 CPU	1 GB
	25% CPUs	20% RAM

Dominant resource of User 1 is CPU (as 25% > 20%)

A user's *dominant share:* fraction of dominant resource she is allocated

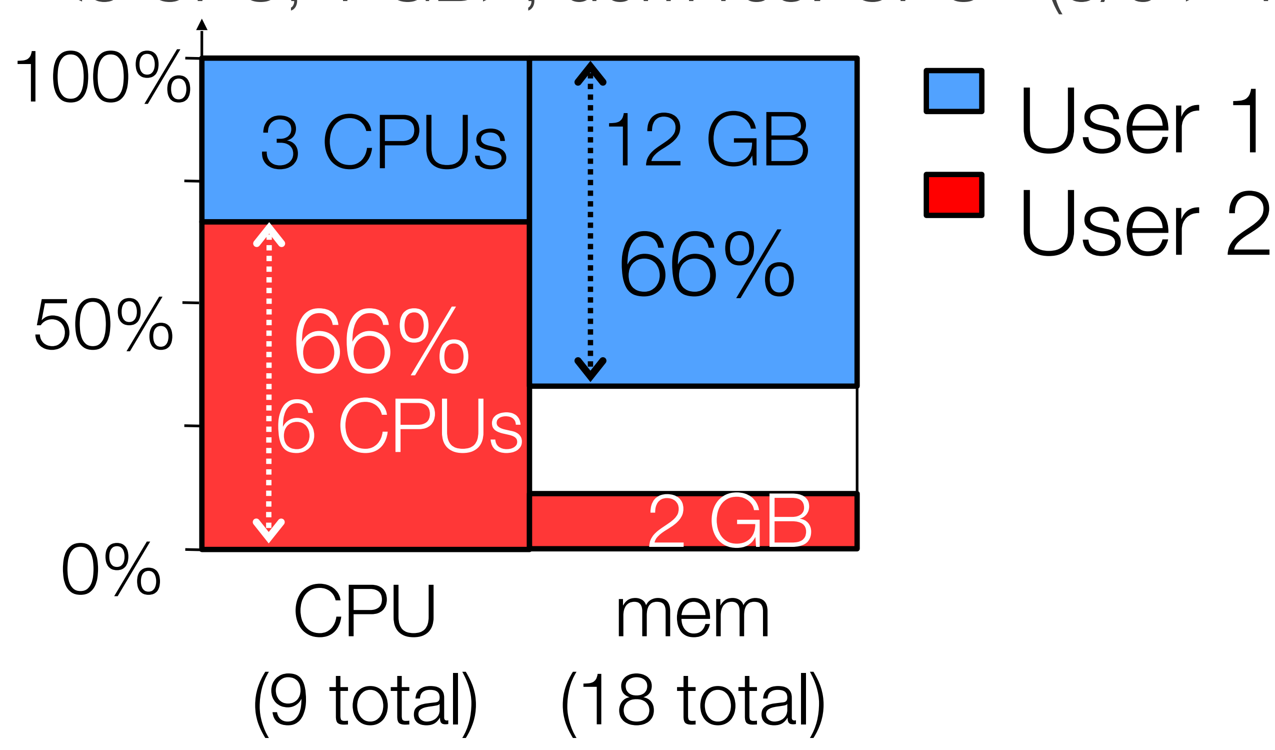
- User 1's dominant share is 25%

Dominant Resource Fairness (DRF)

Apply max-min fairness to dominant shares

Equalize the dominant share of the users. Example:

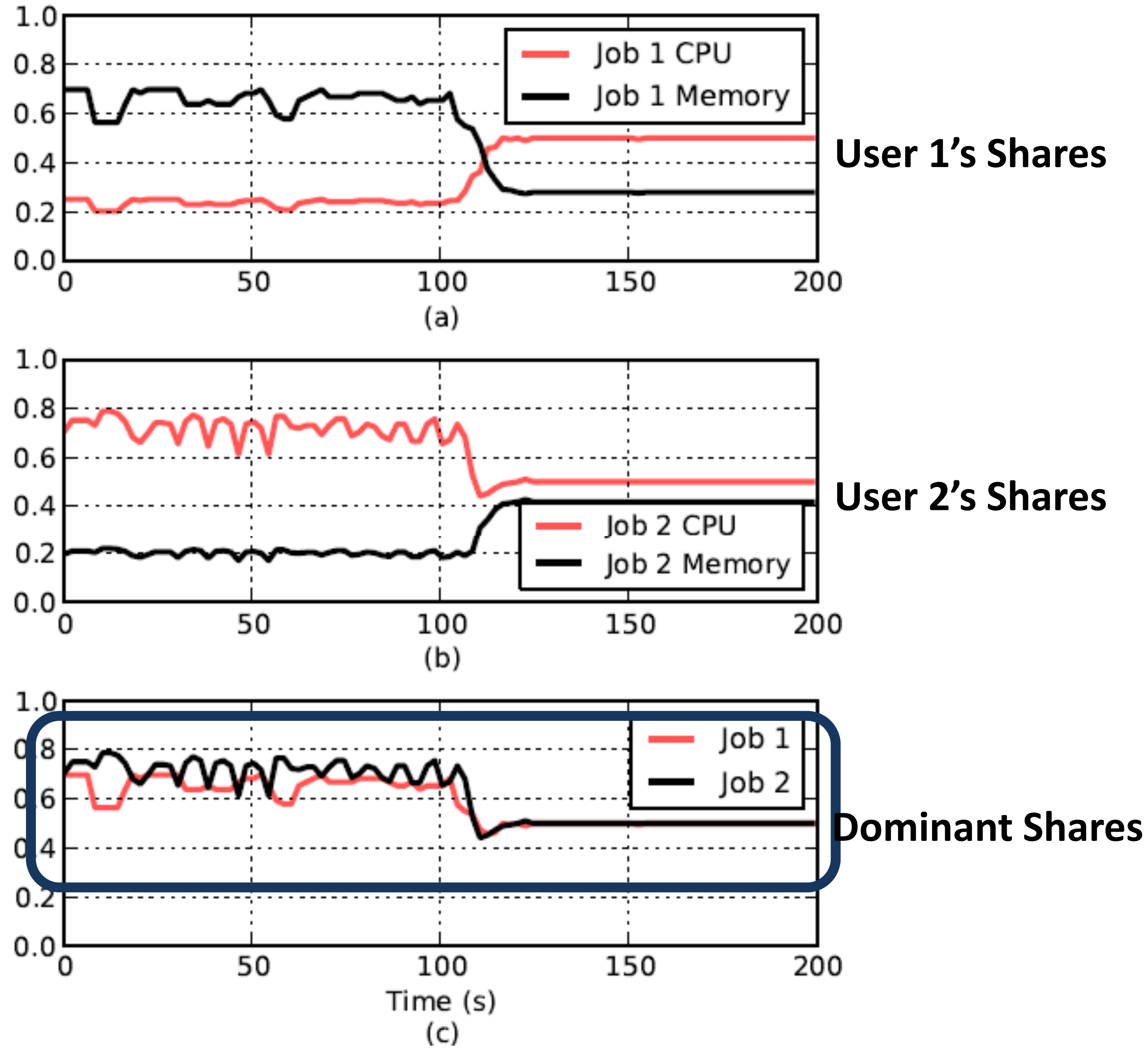
- Total resources: <9 CPU, 18 GB>
- User 1 demand: <1 CPU, 4 GB>; dom res: mem (1/9 < 4/18)
- User 2 demand: <3 CPU, 1 GB>; dom res: CPU (3/9 > 1/18)



Online DRF Scheduler

Whenever there are available resources and tasks to run:
Schedule a task to the user with smallest dominant share

DRF inside Mesos on EC2



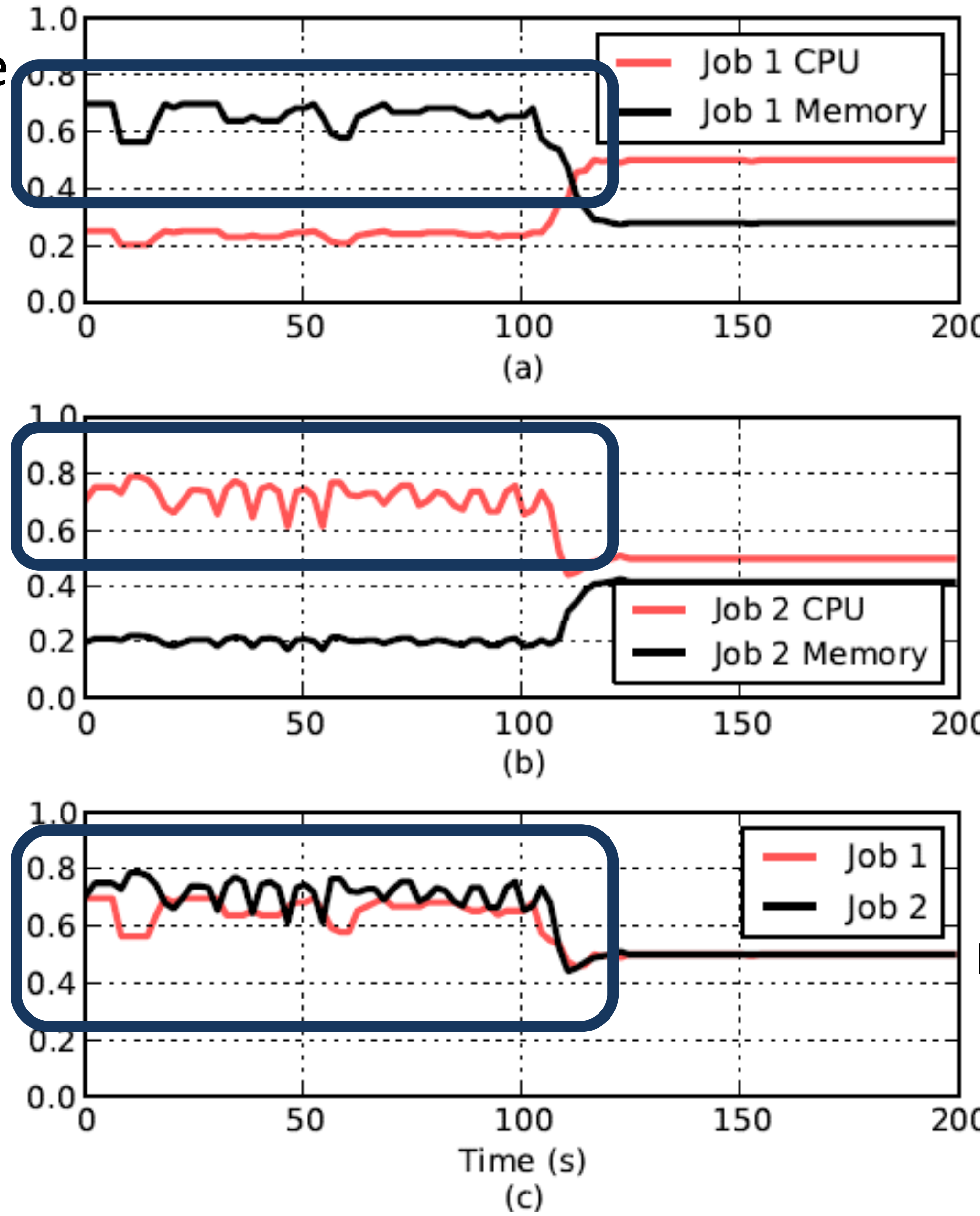
Dominant shares are equalized

DRF inside Mesos on EC2

Dominant resource is memory

Dominant resource is CPU

Share guarantee: ~70% dominant share



User 1's Shares

User 2's Shares

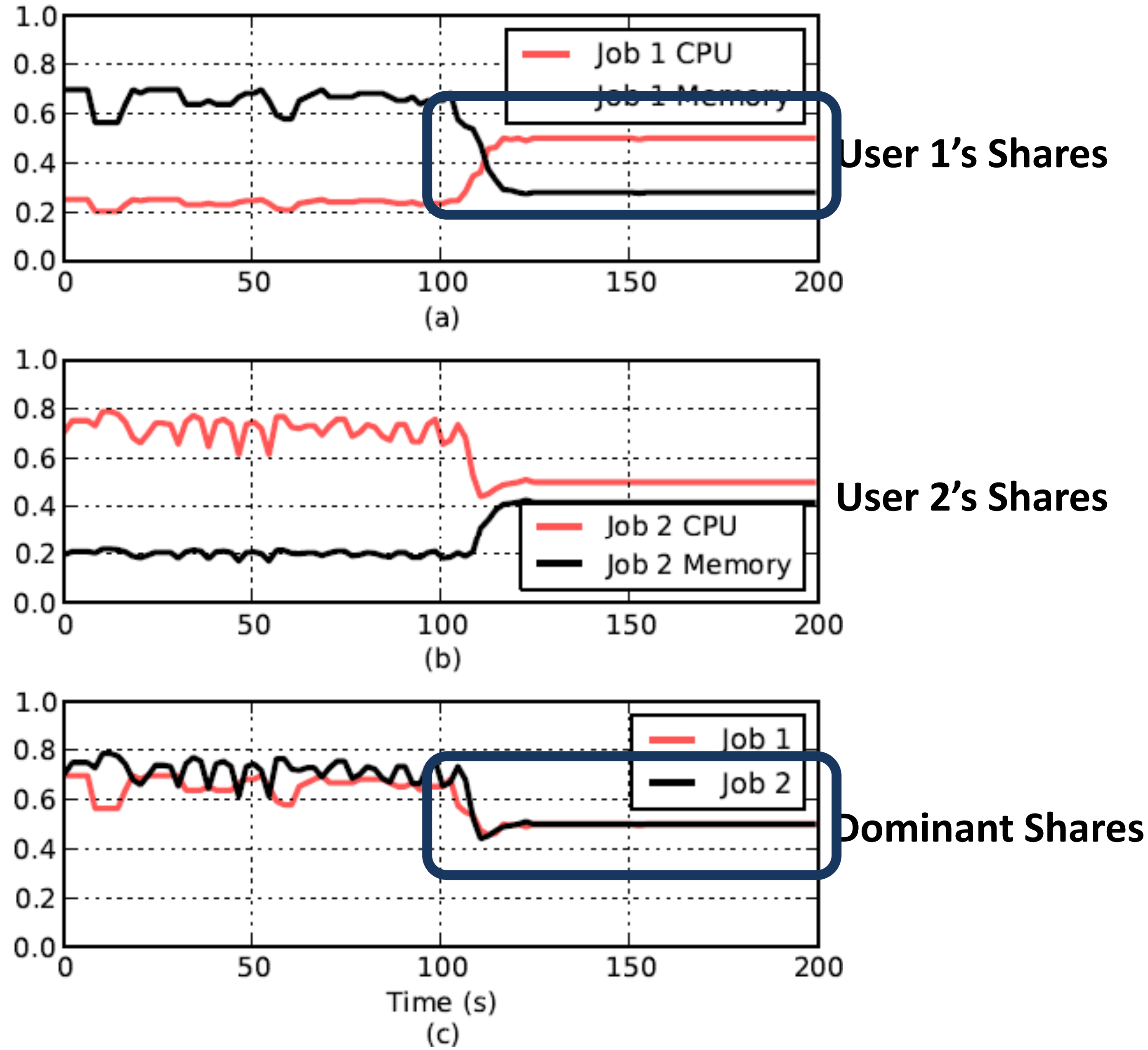
Dominant Shares

DRF inside Mesos on EC2

Dominant resource is CPU

Dominant resource is CPU

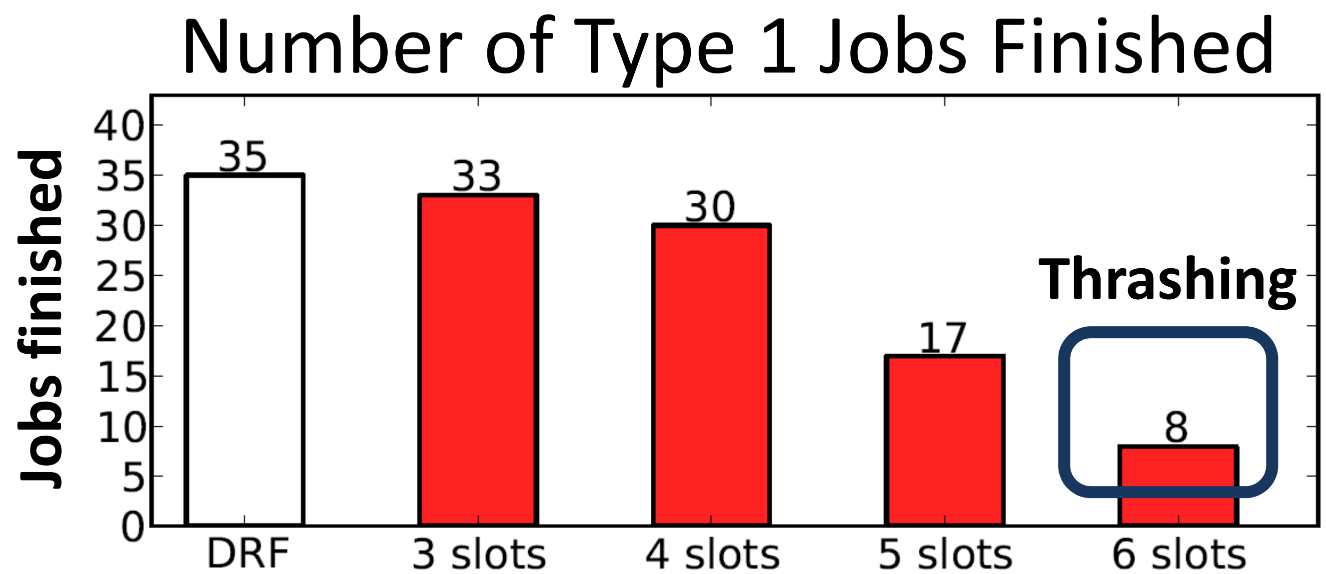
Share guarantee: ~50% dominant share

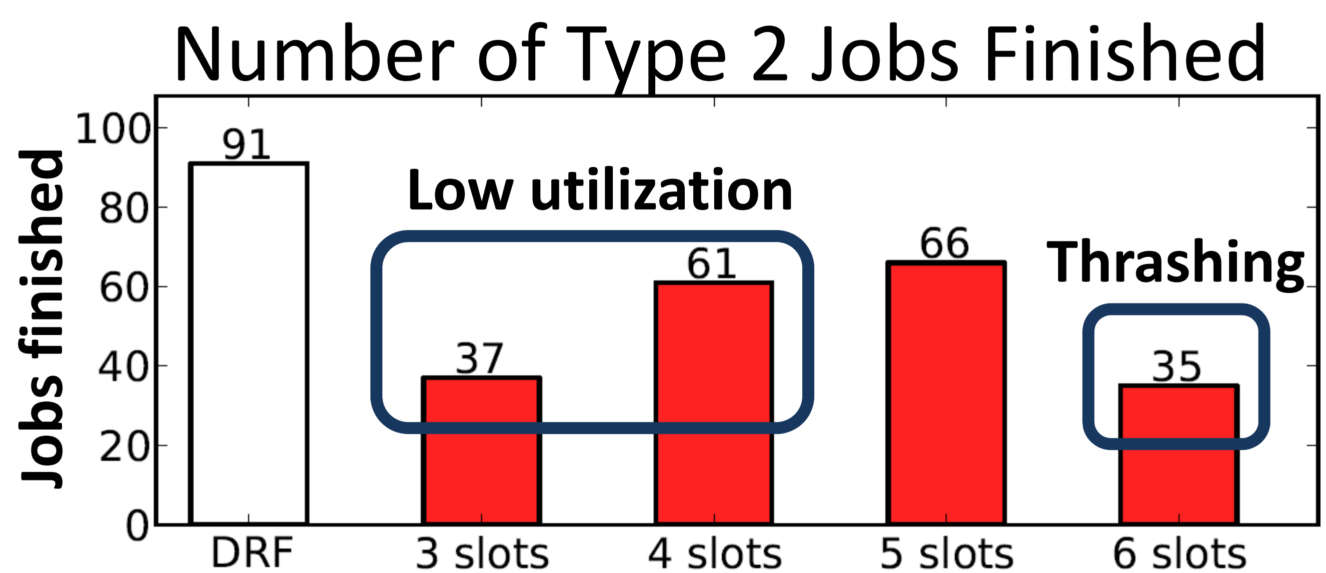


How is fairness solved in datacenters today?

- Hadoop Fair Scheduler/capacity/Quincy
  - Each machine consists of k slots (e.g. k=14)
  - Run at most one task per slot
  - Give jobs "equal" number of slots, i.e., apply max-min fairness to slot-count
- This is what we compare against

Experiment: DRF vs Slots

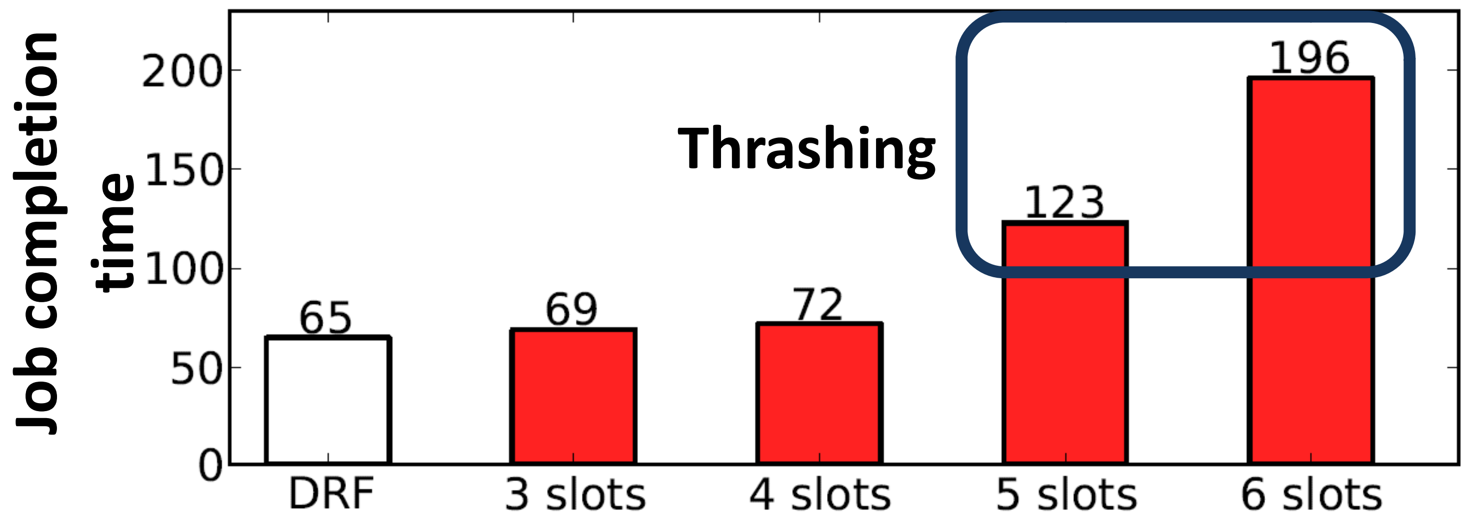




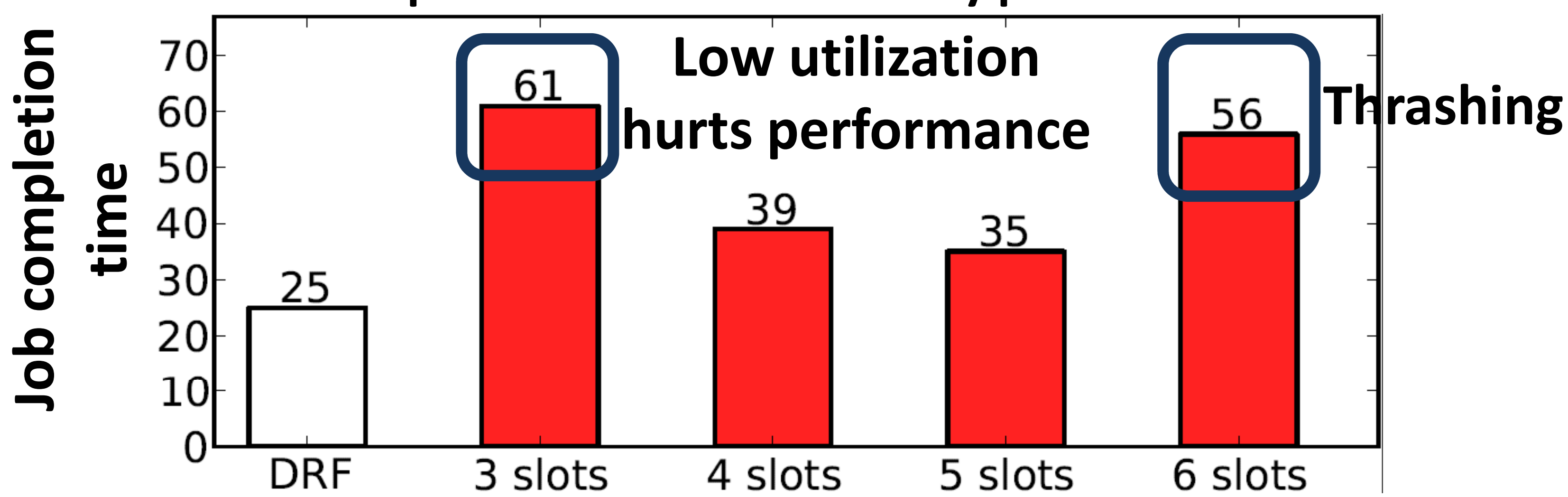
Type 1 jobs <2 CPU, 2 GB> Type 2 jobs <1 CPU, 0.5GB>

Experiment: DRF vs Slots

Completion Time of Type 1 Jobs



Completion Time of Type 2 Jobs



Type 1 job <2 CPU, 2 GB> **Type 2 job <1 CPU, 0.5GB>**

Reduction in Job Completion Time DRF vs slots

- Simulation of 1-week Facebook traces