Dominant Resource Fairness (DRF)

Fair Allocation of Multiple Resource Types

Ali Ghodsi, Matei Zaharia
Benjamin Hindman, Andy Konwinski,
Scott Shenker, Ion Stoica

University of California, Berkeley

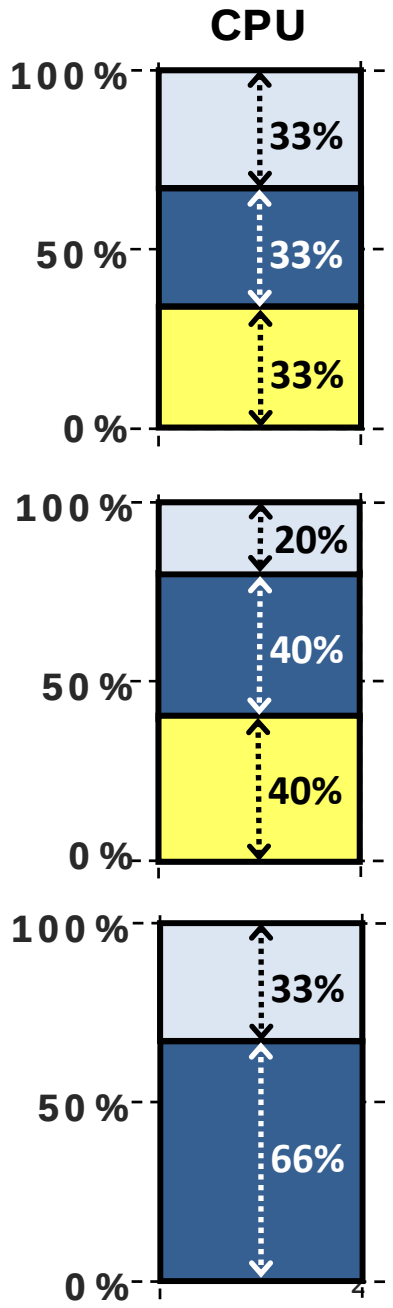
alig@cs.berkeley.edu

What is fair sharing?

- n users want to share a resource (e.g. CPU)
  - Solution:
    Allocate 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



Properties of max-min fairness

- Share guarantee
  - Each user can get 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
- Max-min fairness is the only ”reasonable” mechanism with these two properties

Why care about fairness?

- Desirable properties of max-min fairness
  - *Isolation policy:*

    A user gets her fair share irrespective of the demands of other users

  - *Flexibility separates mechanism from policy:*

    Proportional sharing, priority, reservation,...

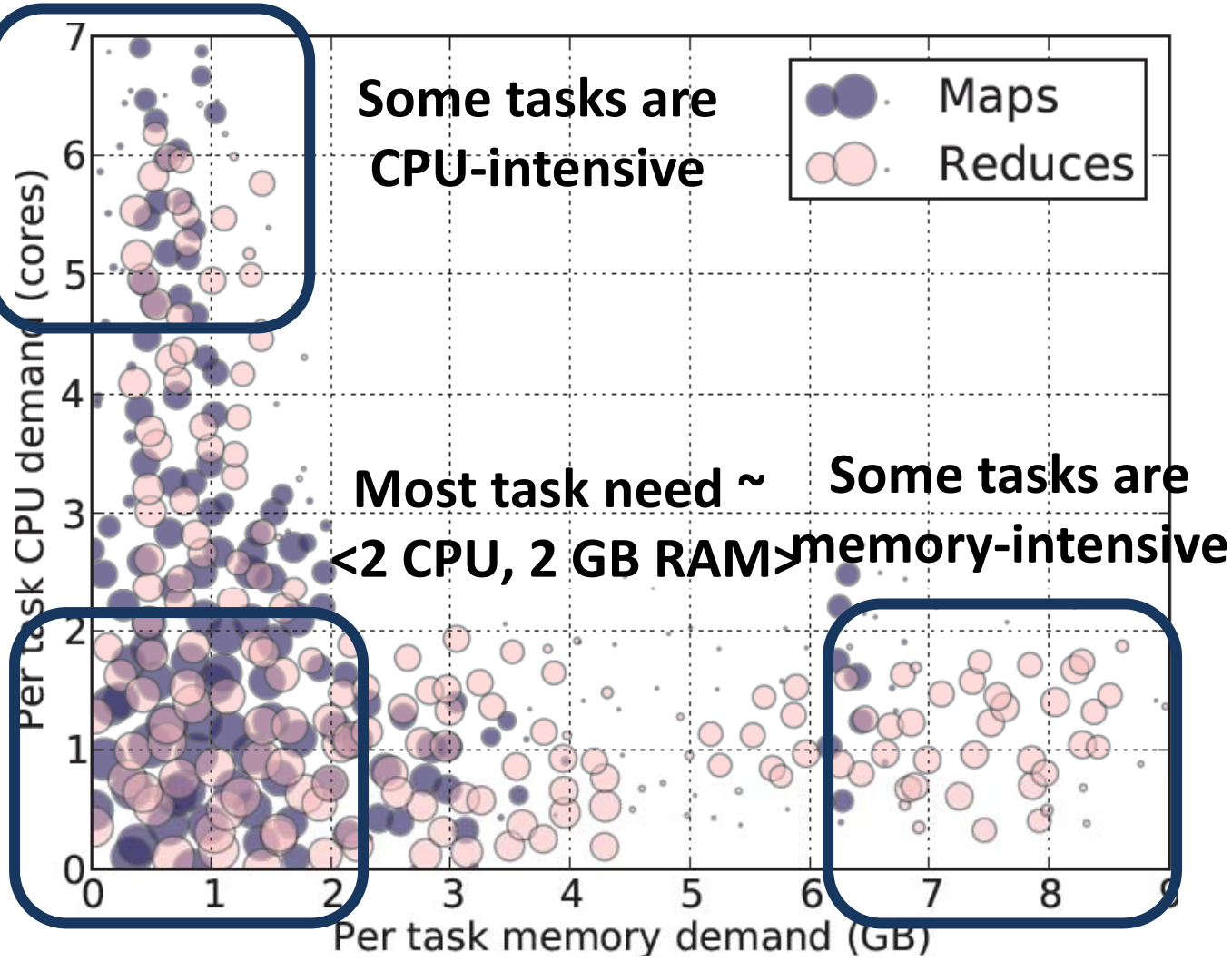
- *Many schedulers* use max-min fairness
  - Datacenters: Hadoop's fair sched, capacity, Quincy
  - OS: rr, prop sharing, lottery, linux cfs, ...
  - Networking: wfq, wf2q, sfq, drr, csfq, ...

Why is max-min fairness not enough?

- Job scheduling in datacenters is not only about CPUs
  - Jobs consume CPU, memory, disk, and I/O

- Does this pose any challenge?

Heterogeneous Resource Demands

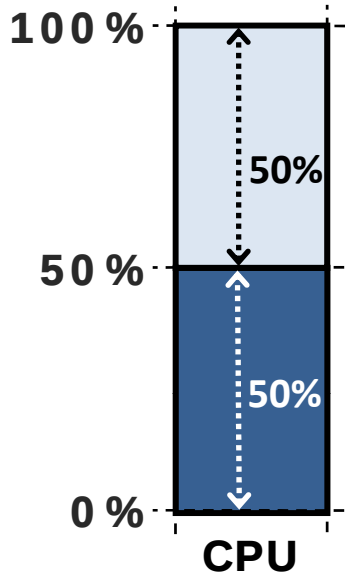


2000-node Hadoop Cluster at Facebook (Oct 2010)

Problem

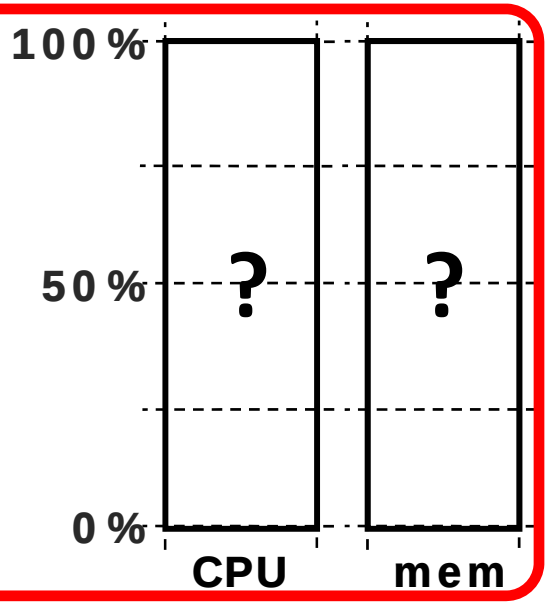
Single resource example

- 1 resource: CPU
- User 1 wants **<1 CPU>** per task
- User 2 wants **<3 CPU>** per task



Multi-resource example

- 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?***



Problem definition

How to fairly share multiple resources when users have heterogenous demands on them?

Talk Outline

- What properties do we want?
- How do we solve it (DRF)?
- How would an economist solve this?
- How well does this work in practice?

Model

- Users have *tasks* according to a *demand vector*
  - e.g. **<2, 3, 1>** user's tasks need 2 R_1, 3 R_2, 1 R_3
  - Not needed in practice, measure actual consumption
- Resources given in multiples of demand vectors
- Assume divisible resources

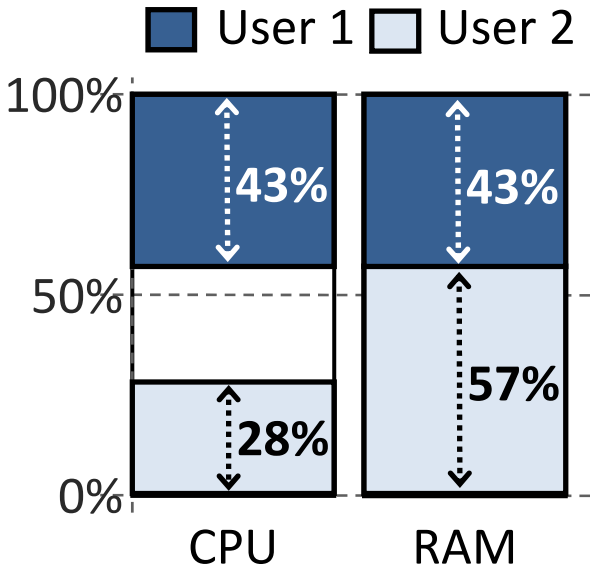
A Natural Policy

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

- Cluster with 70 CPUs, 70 GB RAM
  - U_1 needs <2 CPU, 2 GB RAM> per task
  - U_2 needs <1 CPU, 2 GB RAM> per task

A Natural Policy

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



Problem

User 1 has < 50% of both CPUs and RAM

Better off in a separate cluster with 50% of the resources

- Asset fairness yields
  - U_1: 15 tasks: 30 CPUs, 30 GB (∑=60)
  - U_2: 20 tasks: 20 CPUs, 40 GB (∑=60)

Share Guarantee

- Every user should get $1/n$ of at least one resource
- Intuition:
  - "You shouldn't be worse off than if you ran your own cluster with 1/n of the resources"

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 utllization**

Strategy-proofness

- A user should not be able to increase her allocation by lying about her demand vector
- Intuition:
  - Users are incentivized to provide truthful resource requirements

Challenge

- Can we find a fair sharing policy that provides
  - Strategy-proofness
  - Share guarantee

- Max-min fairness for a single resource had these properties
  - Can we generalize max-min fairness to multiple resources?

Talk Outline

- What properties do we want?
- How do we solve it (DRF)?
- How would an economist solve this?
- How well does this work in practice?

Dominant Resource Fairness

- A user's *dominant resource* is the resource she has the biggest share of
  - Example:

    Total resources: **<10 CPU, 4 GB>**

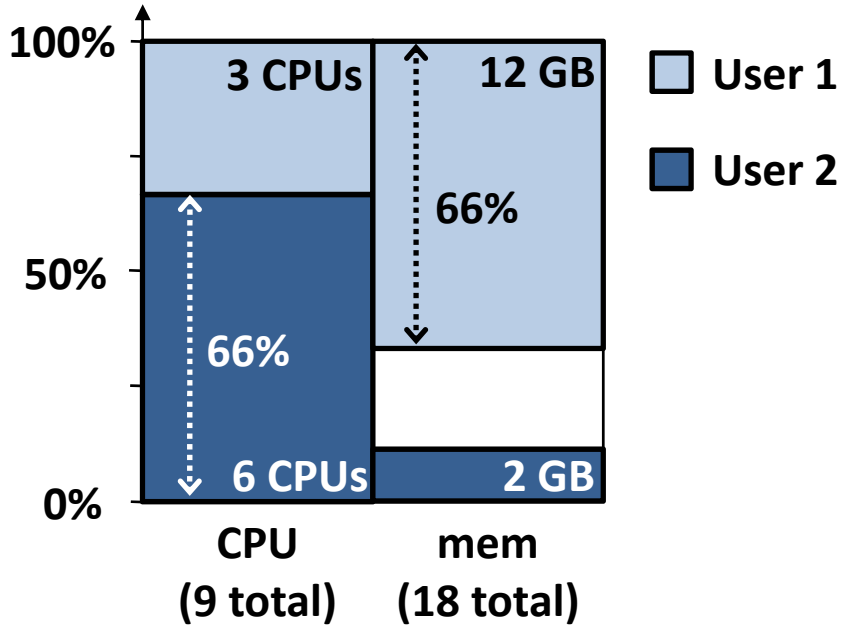
    User 1's allocation: **<2 CPU, 1 GB>**

    Dominant resource is memory as 1/4 > 2/10 (1/5)

- A user's *dominant share* is the fraction of the dominant resource she is allocated
  - User 1's dominant share is **25%** (1/4)

Dominant Resource Fairness (2)

- ***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**
    User 2 demand: **<3 CPU, 1 GB>** dom res: **CPU**



Online DRF Scheduler

Whenever there are available resources and tasks to run:

Schedule a task to the user with smallest dominant share

- O(log *n*) time per decision using binary heaps

Talk Outline

- What properties do we want?
- How do we solve it (DRF)?
- How would an economist solve this?
- How well does this work in practice?

Why not use pricing?

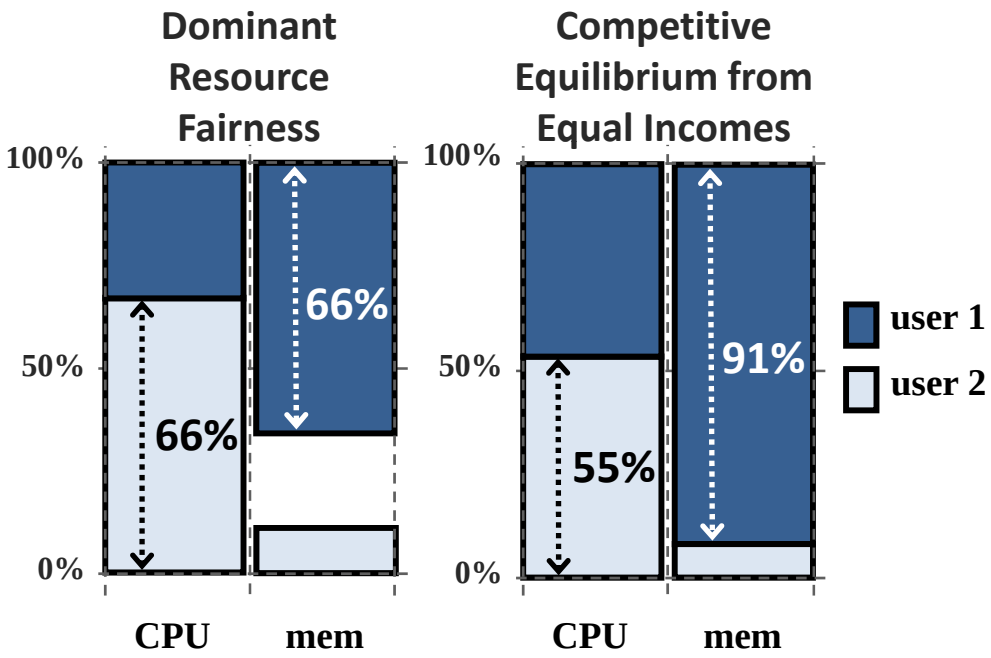
- Approach
  - Set prices for each good
  - Let users buy what they want
- Problem
  - How do we determine the right prices for different goods?

How would an economist solve it?

- Let the market determine the prices
- *Competitive Equilibrium from Equal Incomes (CEEI)*
  - Give each user 1/n of every resource
  - Let users trade in a perfectly competitive market
- **Not strategy-proof!**

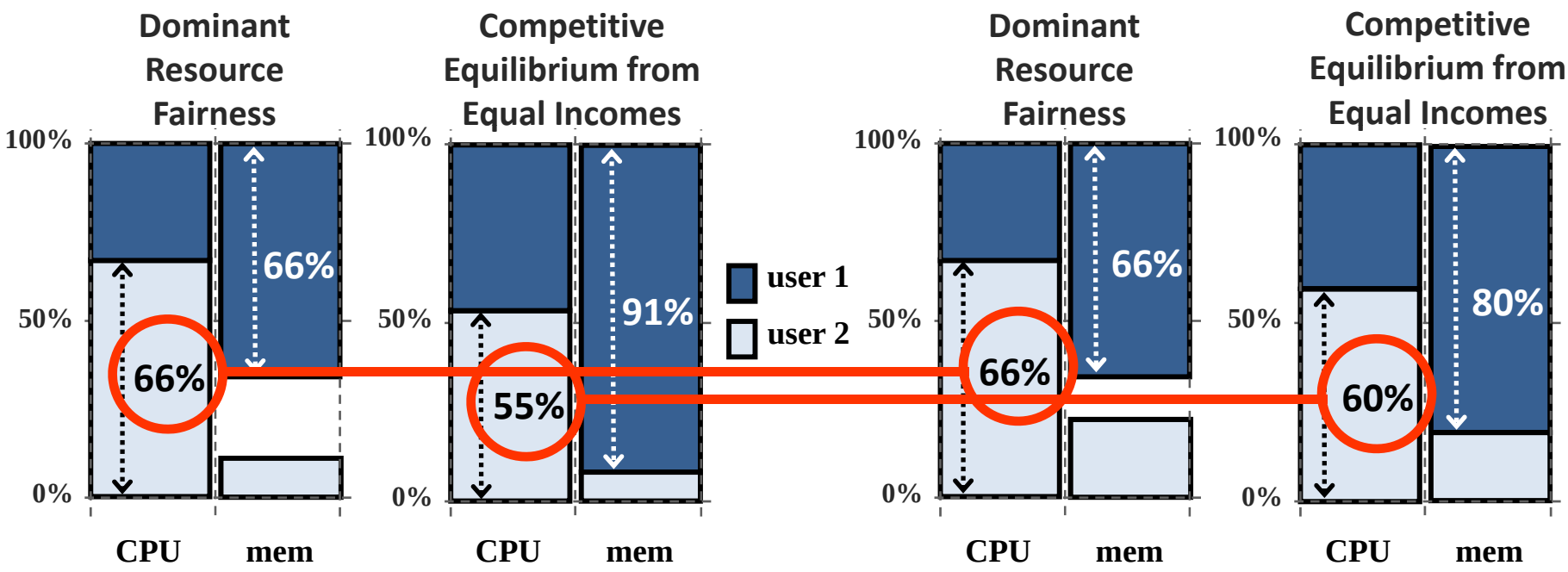
DRF vs CEEI

- User 1: <1 CPU, 4 GB> User 2: <3 CPU, 1 GB>
  - DRF more fair, CEEI better utilization



DRF vs CEEI

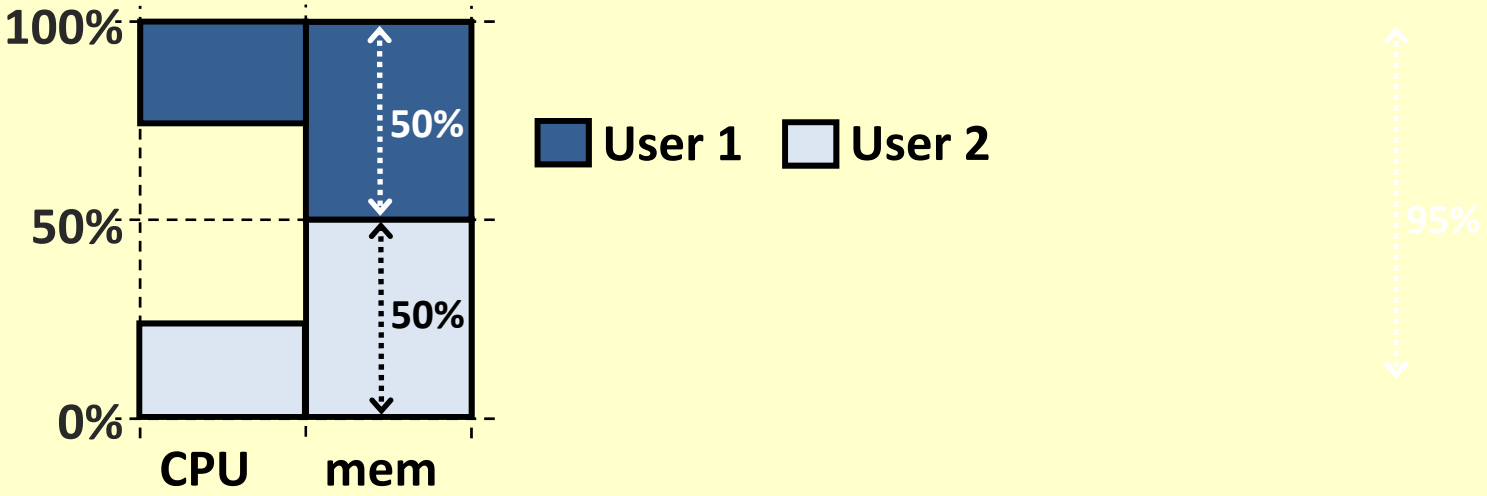
- User 1: <1 CPU, 4 GB> User 2: <3 CPU, 1 GB>
  - DRF more fair, CEEI better utilization



- User 1: <1 CPU, 4 GB> User 2: <3 CPU, 2 GB>
  - User 2 increased her share of both CPU and memory

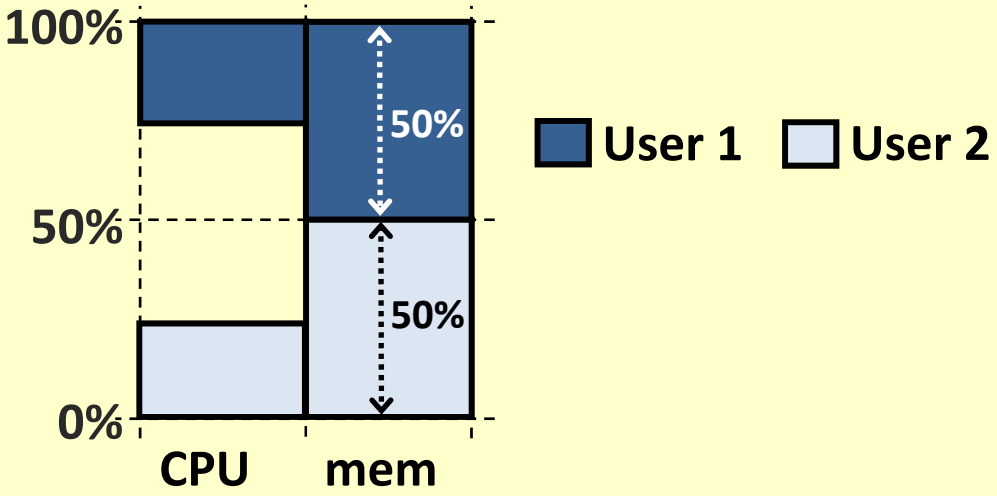
Gaming Utilization-Optimal Schedulers

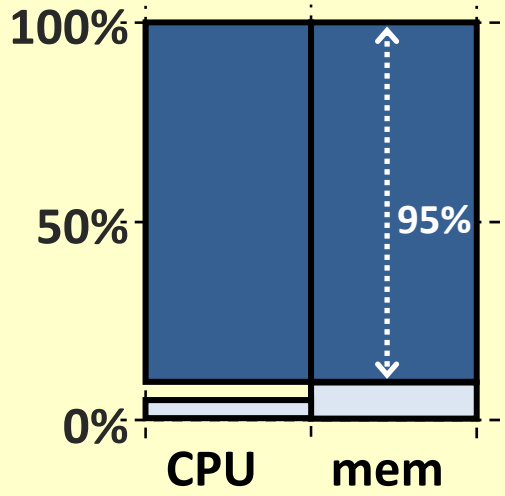
- Cluster with **<100 CPU, 100 GB>**
- 2 users, each demanding **<1 CPU, 2 GB>** per task



Gaming Utilization-Optimal Schedulers

- Cluster with **<100 CPU, 100 GB>**
- 2 users, each demanding **<1 CPU, 2 GB>** per task

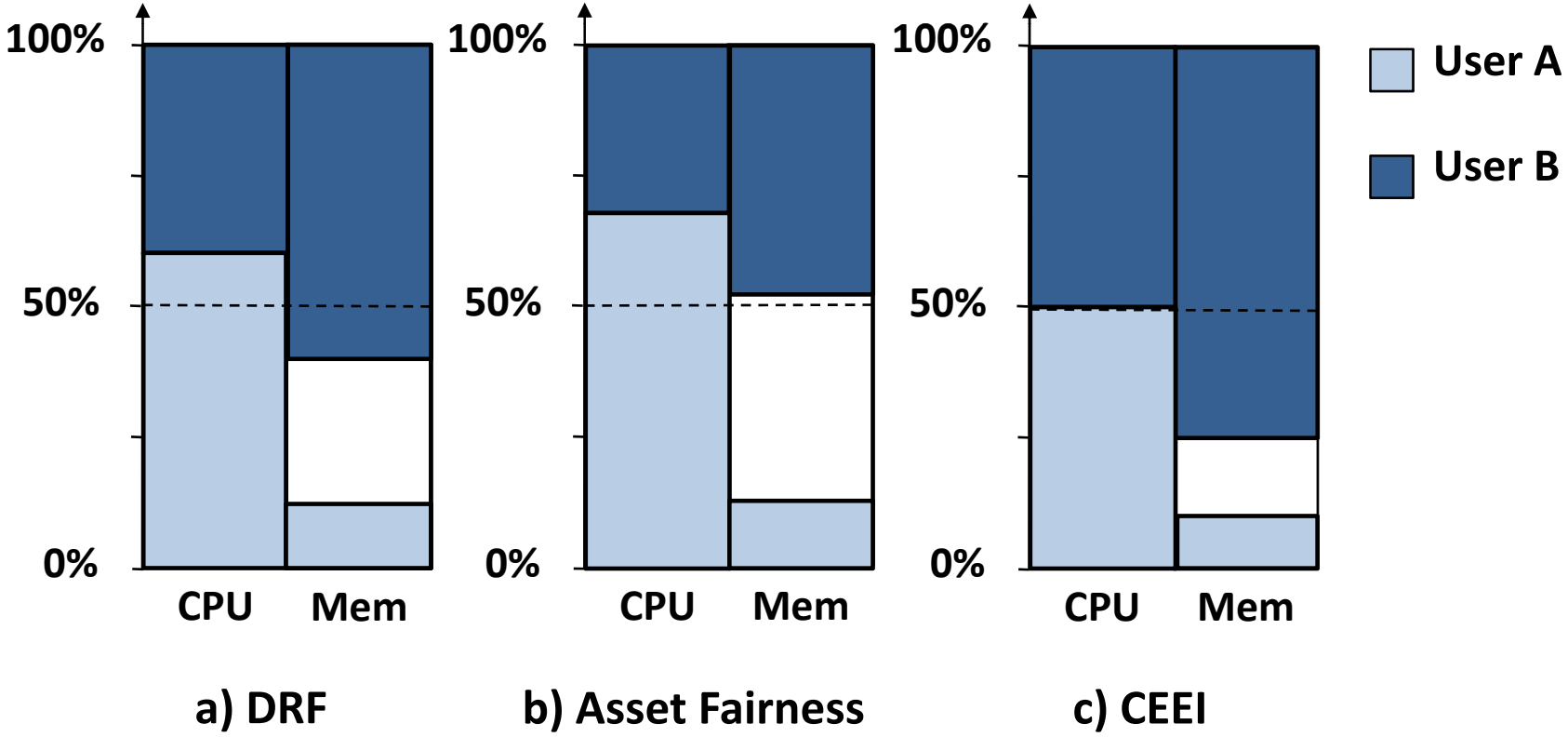




- User 1 lies and demands **<2 CPU, 2 GB>**
- Utilization-Optimal scheduler prefers user 1

Example of DRF vs Asset vs CEEI

- Resources **<1000 CPUs, 1000 GB>**
- 2 users A: **<2 CPU, 3 GB>** and B: **<5 CPU, 1 GB>**



Properties of Policies

Property	Asset	CEEI	DRF
Share guarantee		✔	✔
Strategy-proofness	✔		✔
Pareto efficiency	✔	✔	✔
Envy-freeness	✔	✔	✔
Single resource fairness	✔	✔	✔
Bottleneck res. fairness		✔	✔
Population monotonicity	✔		✔
Resource monotonicity			

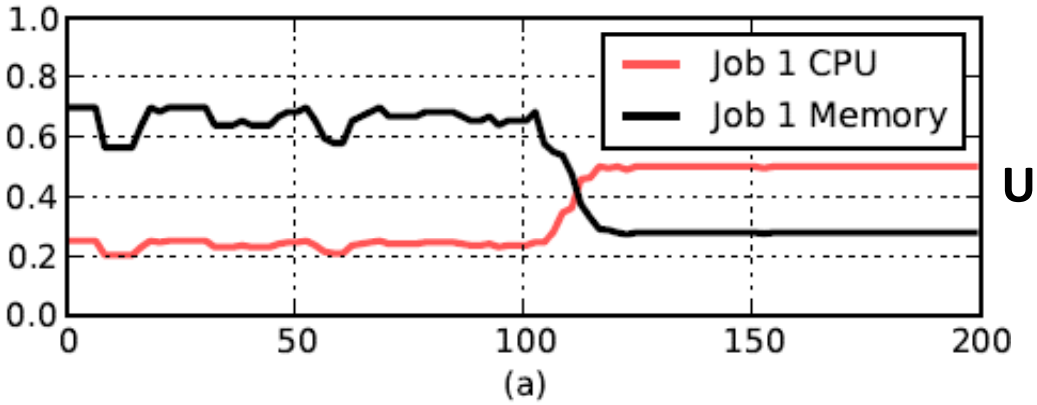
Talk Outline

- What properties do we want?
- How do we solve it (DRF)?
- How would an economist solve this?
- How well does this work in practice?

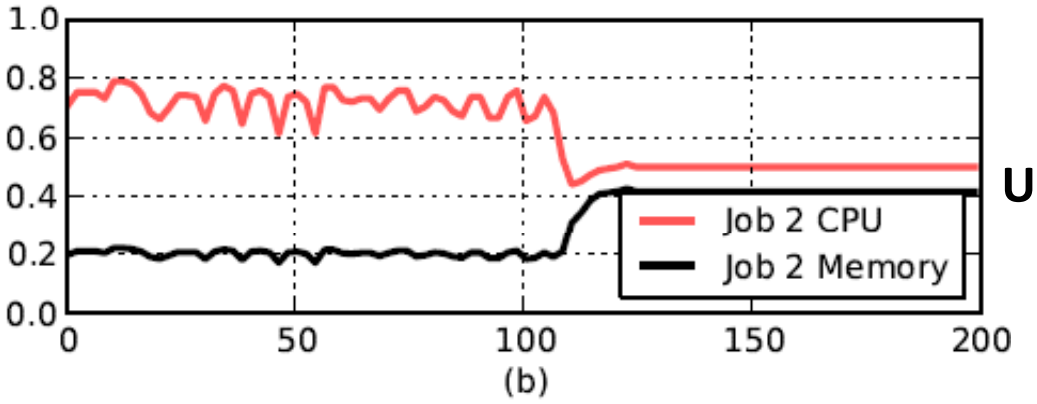
Evaluation Methodology

- Micro-experiments on EC2
  - Evaluate DRF's dynamic behavior when demands change
  - Compare DRF with current Hadoop scheduler
- Macro-benchmark through simulations
  - Simulate Facebook trace with DRF and current Hadoop scheduler

DRF inside Mesos on EC2

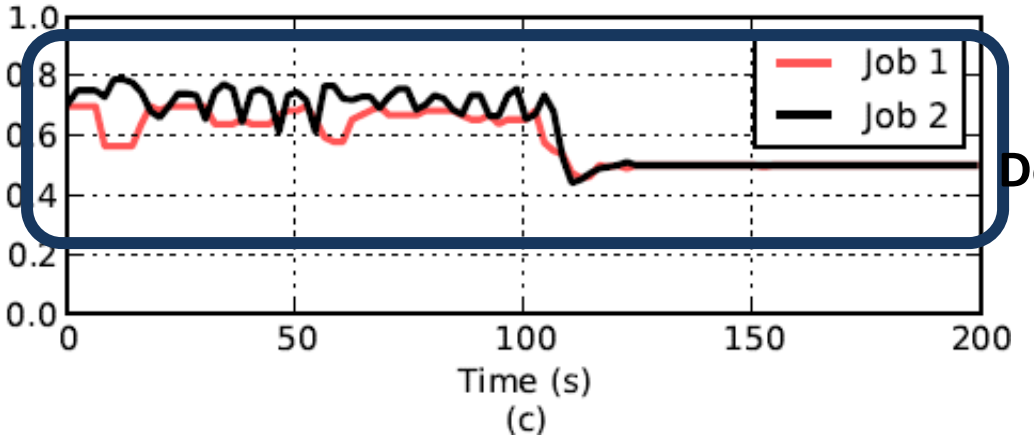


User 1's Shares



User 2's Shares

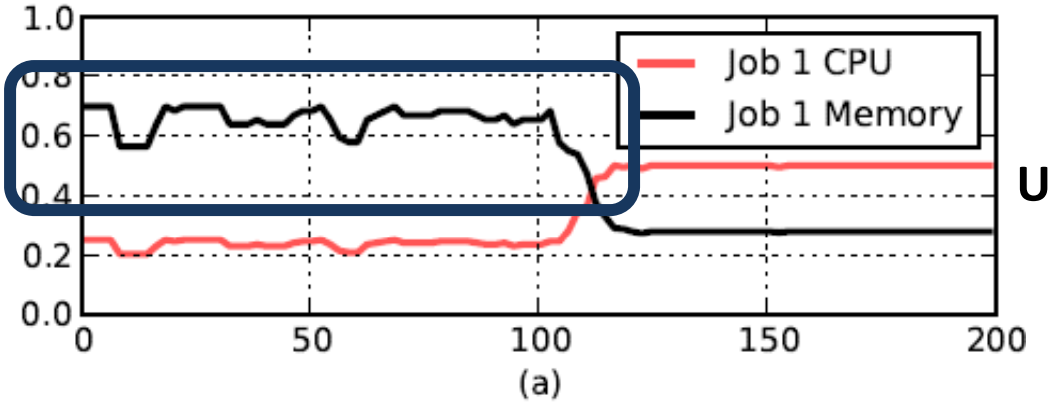
Dominant shares are equalized



Dominant Shares

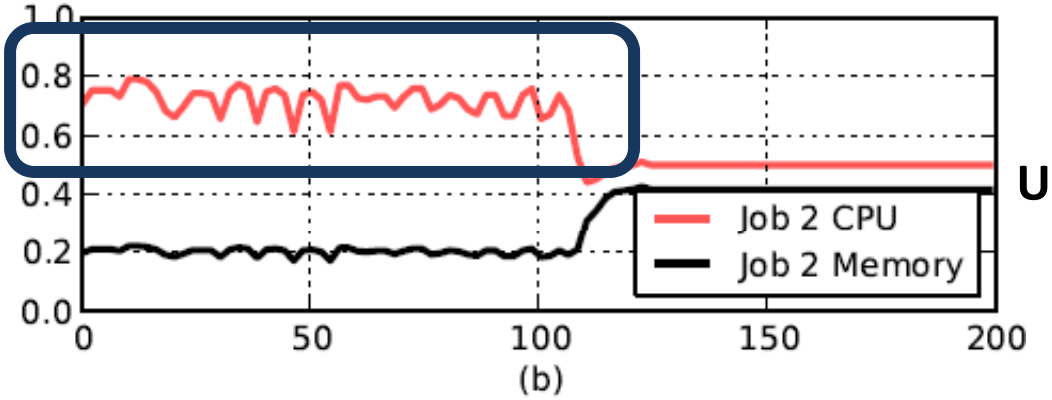
DRF inside Mesos on EC2

Dominant resource is memory



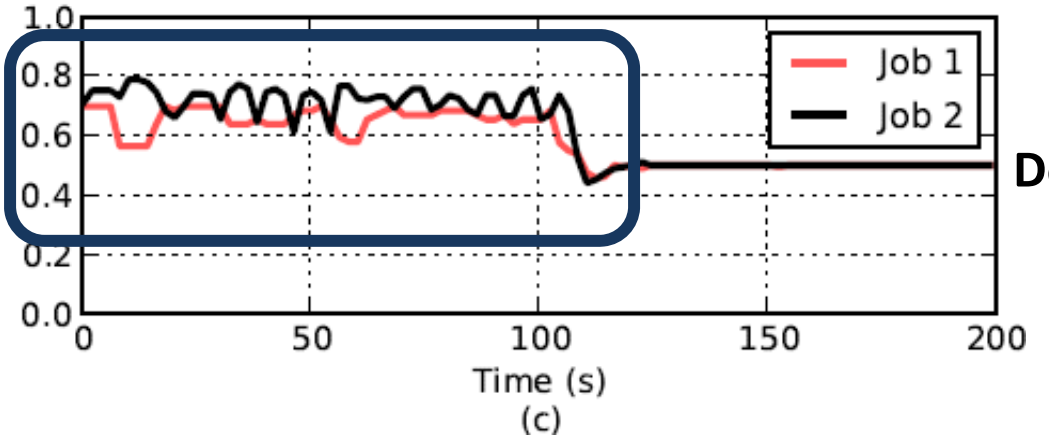
User 1's Shares

Dominant resource is CPU



User 2's Shares

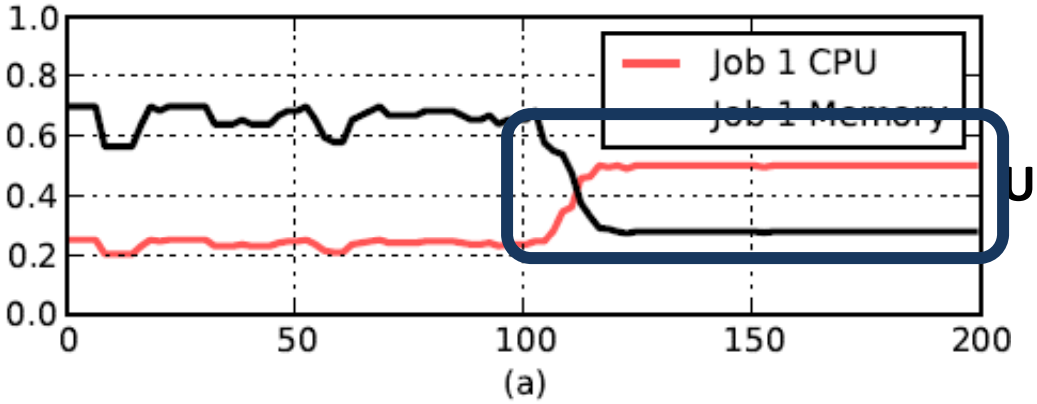
Share guarantee: ~70% dominant share



Dominant Shares

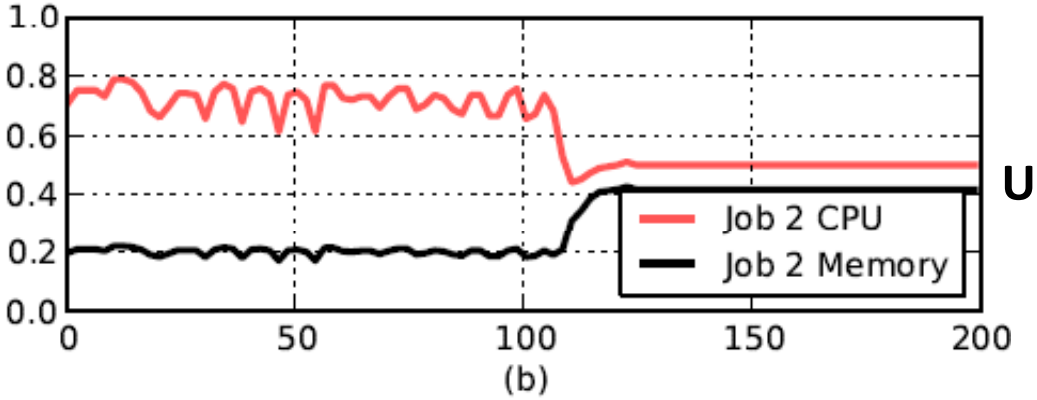
DRF inside Mesos on EC2

Dominant resource is CPU



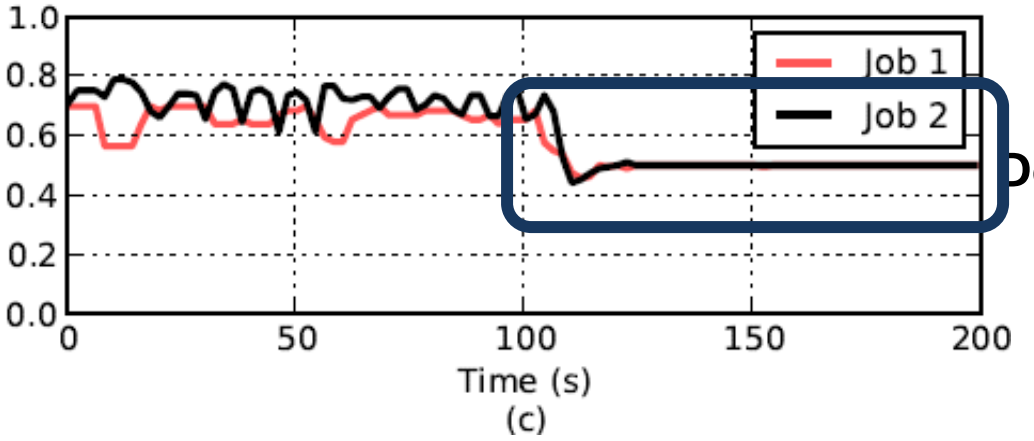
User 1's Shares

Dominant resource is CPU



User 2's Shares

Share guarantee: ~50% dominant share



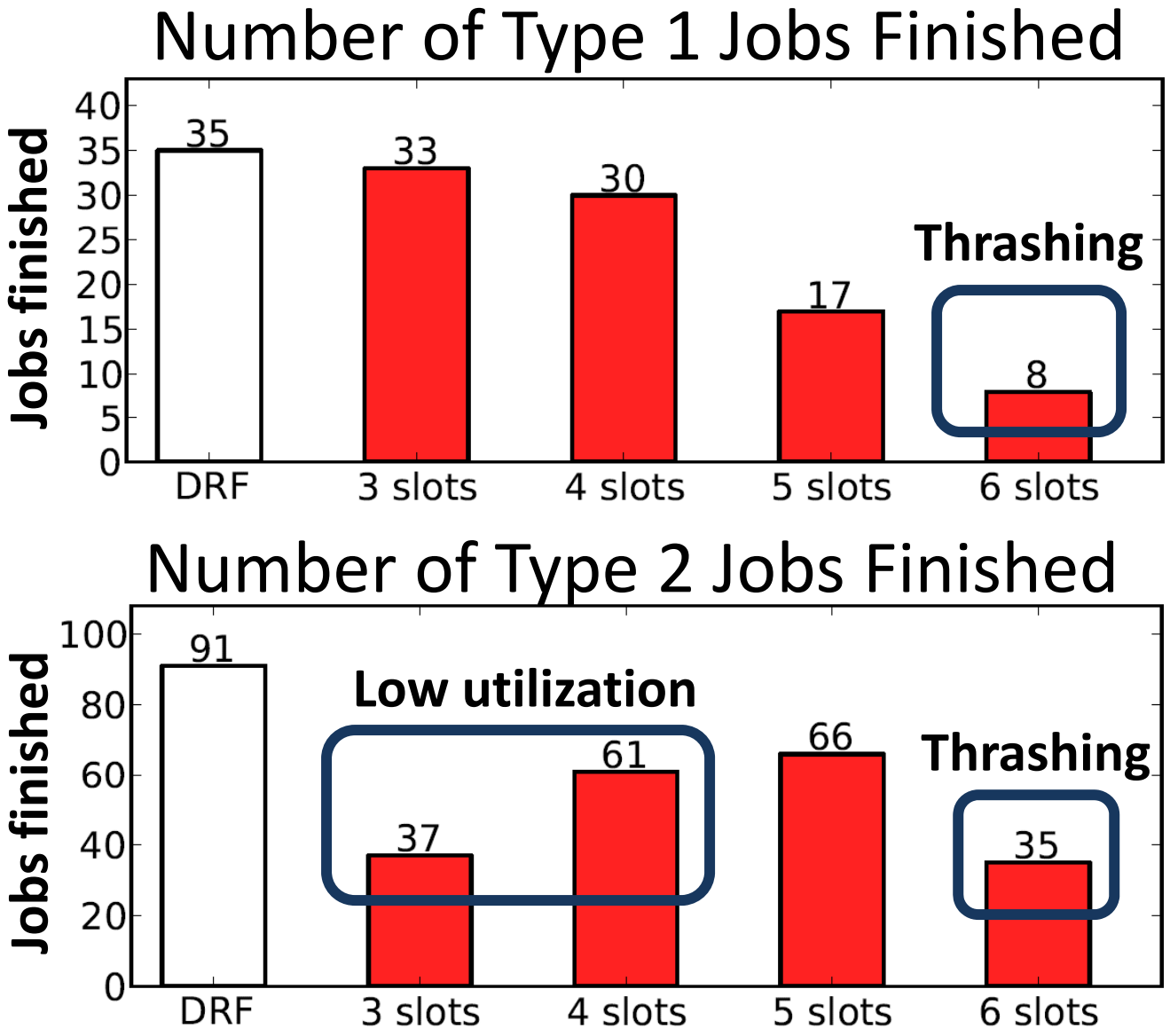
Dominant Shares

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

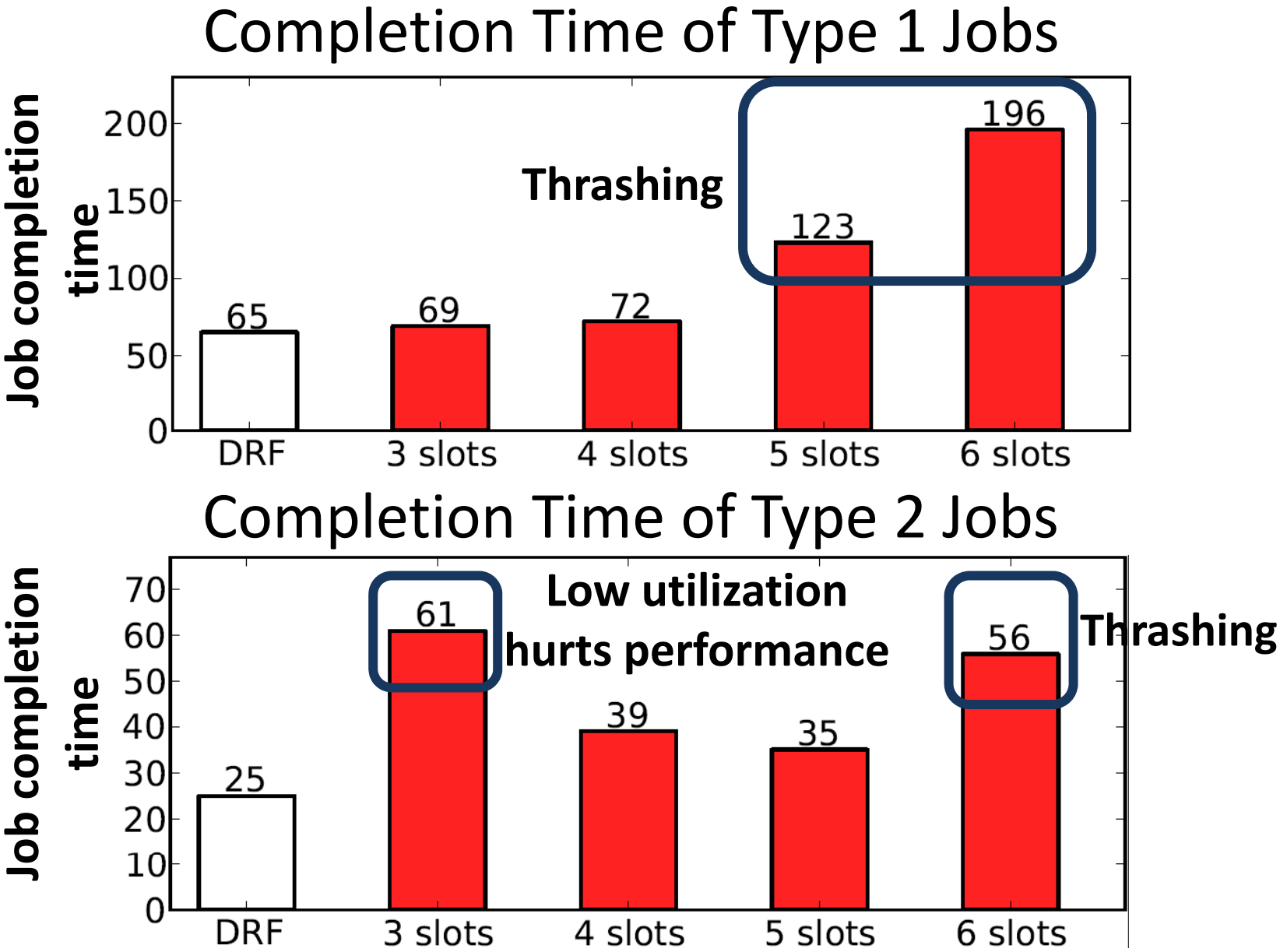
- This is what we compare against

Experiment: DRF vs Slots



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

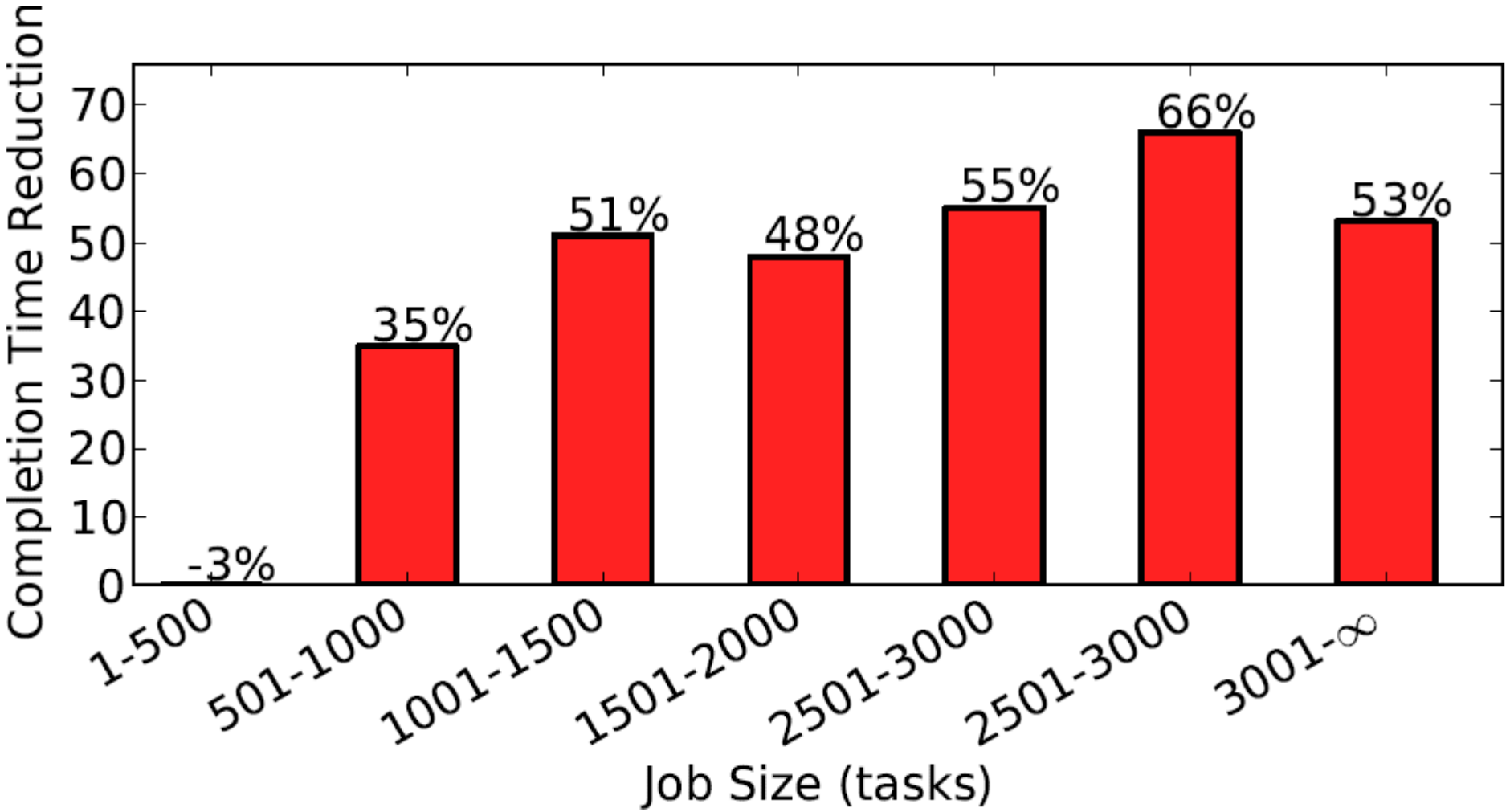
Experiment: DRF vs Slots



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

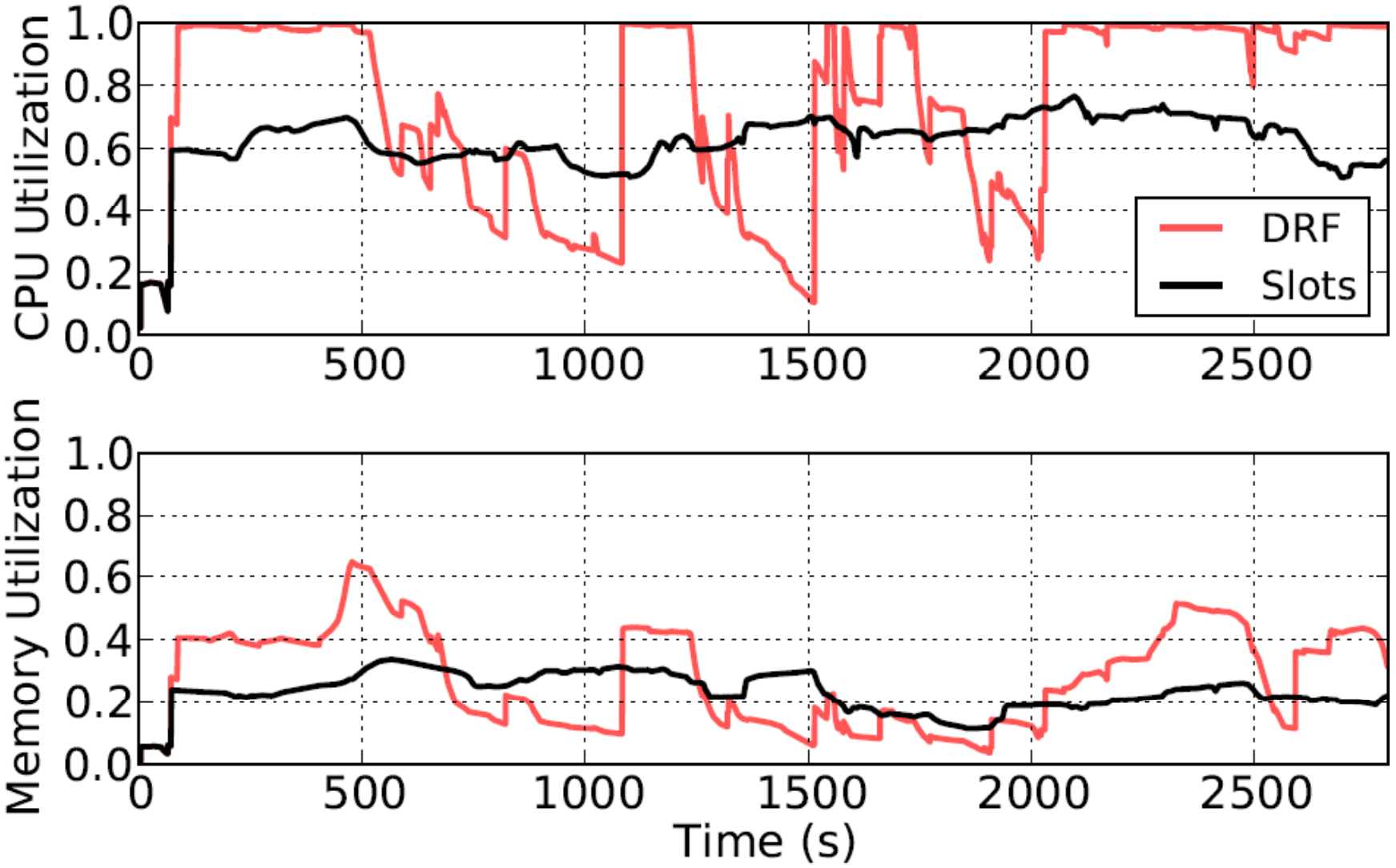
Reduction in Job Completion Time DRF vs slots

- Simulation of 1-week Facebook traces



Utilization of DRF vs slots

- Simulation of Facebook workload



Conclusion

- DRF provides *multiple-resource fairness* in the presence of *heterogenous demand*
  - First generalization of max-min fairness to multiple-resources

- DRF's properties
  - Share guarantee, at least 1/n of one resource
  - Strategy-proofness, lying can only hurt you
  - Performs better than current approaches

Conjecture

- DRF is the *only* "reasonable" policy that satisfies
  - Strategy-proofness
  - Share guarantee

Future Work

- How to pack tasks to get high utilization
- Use DRF as a OS scheduler
- DRF with placement constraints

How do we know the demand vectors?

- They can be *measured*
  - Look at actual resource consumption of a user
- They can be *provided* the by user
  - What is done today
- In both cases, strategy-proofness incentivizes user to consume resources wisely

References

- [Gree09] A. Greenberg, J. Hamilton, D. Maltz, P. Patel, ”The Cost of a Cloud: Research Problems in Data Center Networks”, Sigcomm CCR 39:1, 2009
- [Bert92] D. Bertsekas, R. Gallager, *”Data Networks”*, Prentice Hall, 1992
- [Varian74] H. Varian, *”Equity, envy, and efficiency”*, Journal of Economic Theory 9(1):63–91, 1974
- [Young94] H. Peyton Young, *”Equity: in theory and practise”*, Princeton University, 1994

Appendix

- *A* user U_i has a *bottleneck resource* R_j in an allocation A iff R_j is saturated and all users using R_j have a smaller (or equal) dominant share than U_i

- *Max/min Theorem for DRF*
  - An allocation A is max/min fair iff every user has a bottleneck resource

Appendix 2

- Recall *max/min fairness* from networking
  - Maximize the bandwidth of the minimum flow [Bert92]

- *Progressive filling (PF) algorithm*
  1. Allocate ε to every flow until some link saturated
  2. Freeze allocation of all flows on saturated link and goto 1

Appendix 3

- *P1. Pareto Efficiency*
  - It should not be possible to allocate more resources to any user without hurting others
- *P2. Single-resource fairness*
  - If there is only one resource, it should be allocated according to max/min fairness
- *P3. Bottleneck fairness*
  - If all users want most of one resource(s), that resource should be shared according to max/min fairness

Appendix C
Desirable Fairness Properties (3)

- Assume *positive demands* ($D_{ij} > 0$ for all i and j)

- DRF will allocate same dominant share to all users
  - As soon as PF saturates a resource, allocation frozen

Appendix C
Datacenter Properties (1)

- *P4. Population Monotonicity*
  - If a user leaves and relinquishes her resources, no other user's allocation should get hurt
  - Can happen each time a job finishes

- CEEI violates population monotonicity

Appendix C
Datacenter Properties (2)

- DRF satisfies population monotonicity
  - Assuming positive demands
  - Intuitively DRF gives the same dominant share to all users, so all users would be hurt contradicting Pareto efficiency

Appendix C
The unreachable

- Resource Monotonicity (RM)
  - If a resource is increased, no user's allocation will decrease

- Impossible to satisfy together with Share Guarante and Pareto Efficiency