Lottery Scheduling and Dominant Resource Fairness (Lecture 24, cs262a)

Ion Stoica,
UC Berkeley
November 7, 2016

Today's Papers

Lottery Scheduling: Flexible Proportional-Share Resource Management,

Carl Waldspurger and William Weihl, OSDI'94

(www.usenix.org/publications/library/proceedings/osdi/full_papers/waldspurger.pdf)

Dominant Resource Fairness: Fair Allocation of Multiple Resource Types

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, Ion Stoica, NSDI'11

(https://www.cs.berkeley.edu/~alig/papers/drf.pdf)

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)

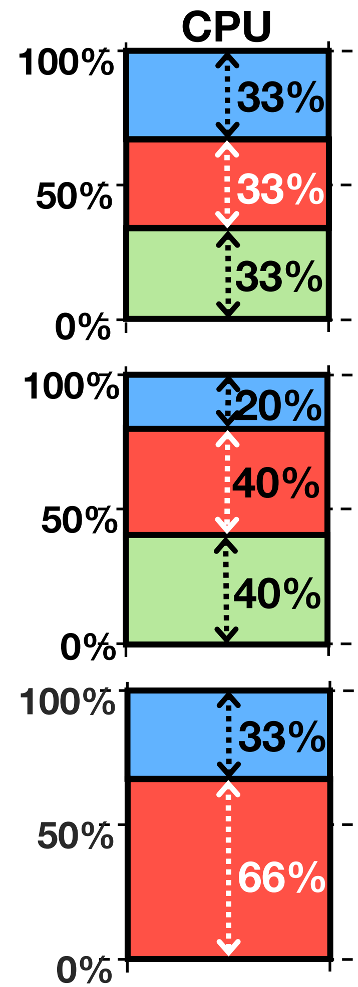
- Solution: give each *1/n* of the shared resource

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

Revervations

- 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

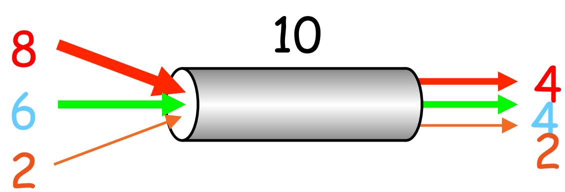
Fair Queueing: Max-min Fairness implementation originated in

- 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]
  - In a fluid flow system it reduces to bit-by-bit round robin
- WFQ in a fluid flow system → Generalized Processor Sharing (GPS) [Parekh & Gallager '92]

Fair Rate Computation

If link congested, compute f such that

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



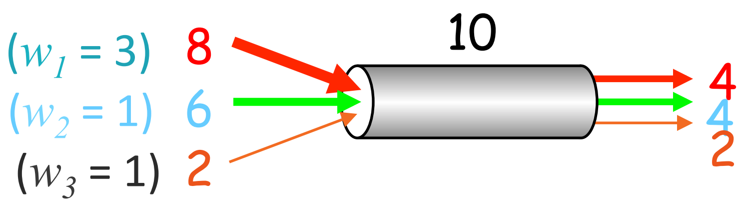
f = 4:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

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]

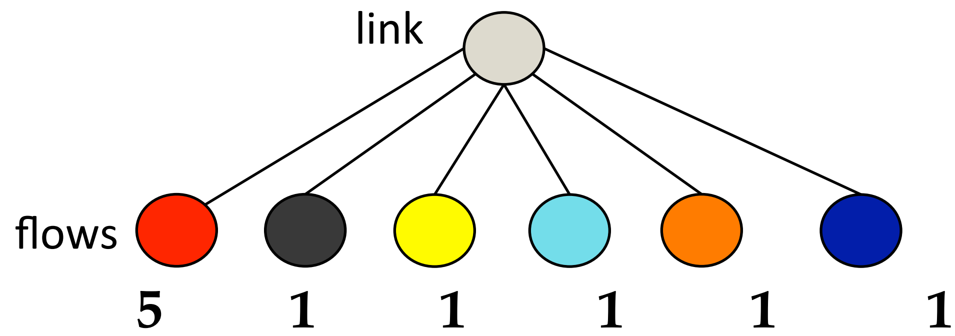
WFQ can be implemented using bit-by-bit weighted round robin in GPS model

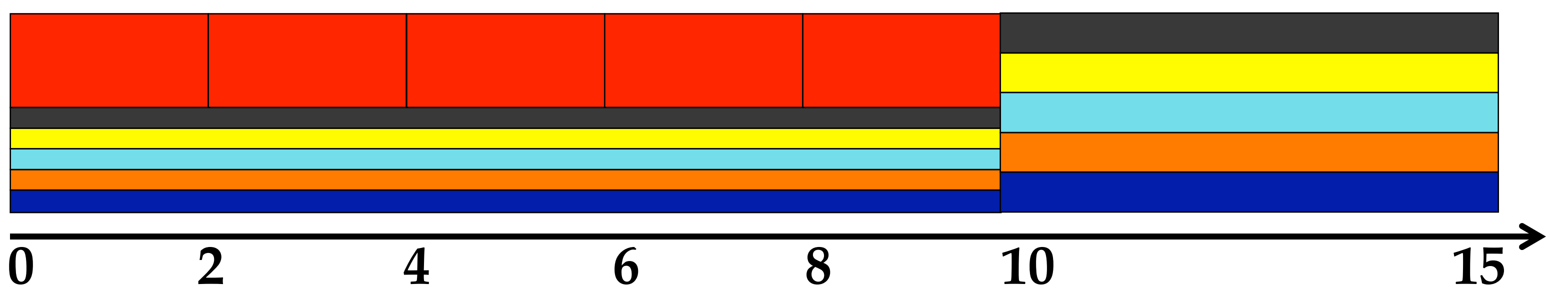
- During each round from each flow that has data to send, send a number of bits equal to the flow’s weight

Generalized Processor Sharing Example

Red session has packets backlogged between time 0 and 10

Other sessions have packets continuously backlogged





Packet Approximation of Fluid System

Standard techniques of approximating fluid GPS

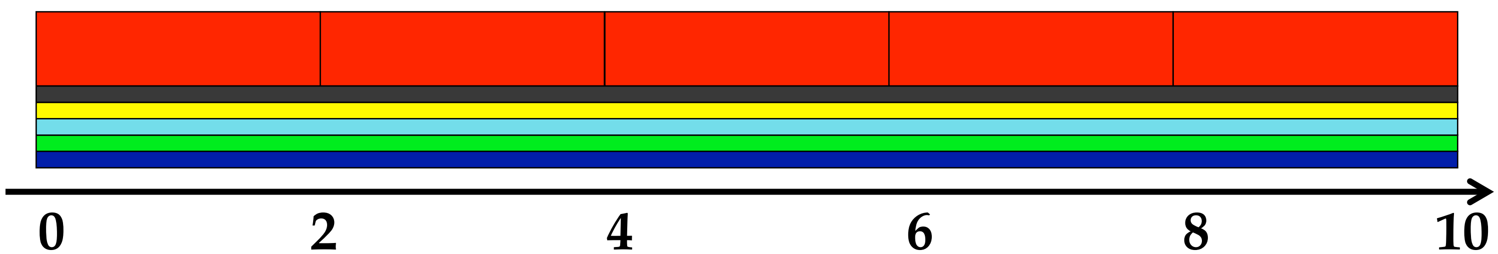
- Select packet that finishes first in GPS assuming that there are no future arrivals

Implementation based on virtual time

- Assign virtual finish time to each packet upon arrival
- Packets served in increasing order of virtual times

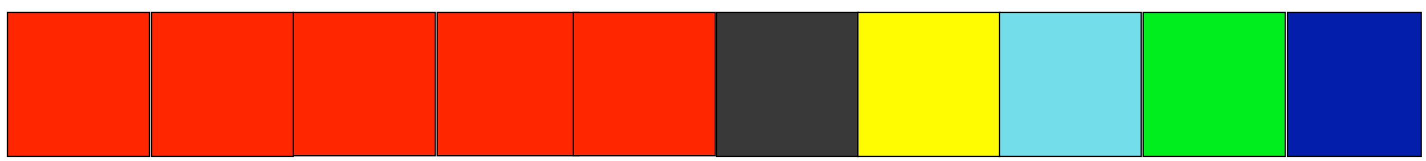
Approximating GPS with WFQ

Fluid GPS system service order



Weighted Fair Queueing

- select the first packet that finishes in GPS



Implementation Challenge

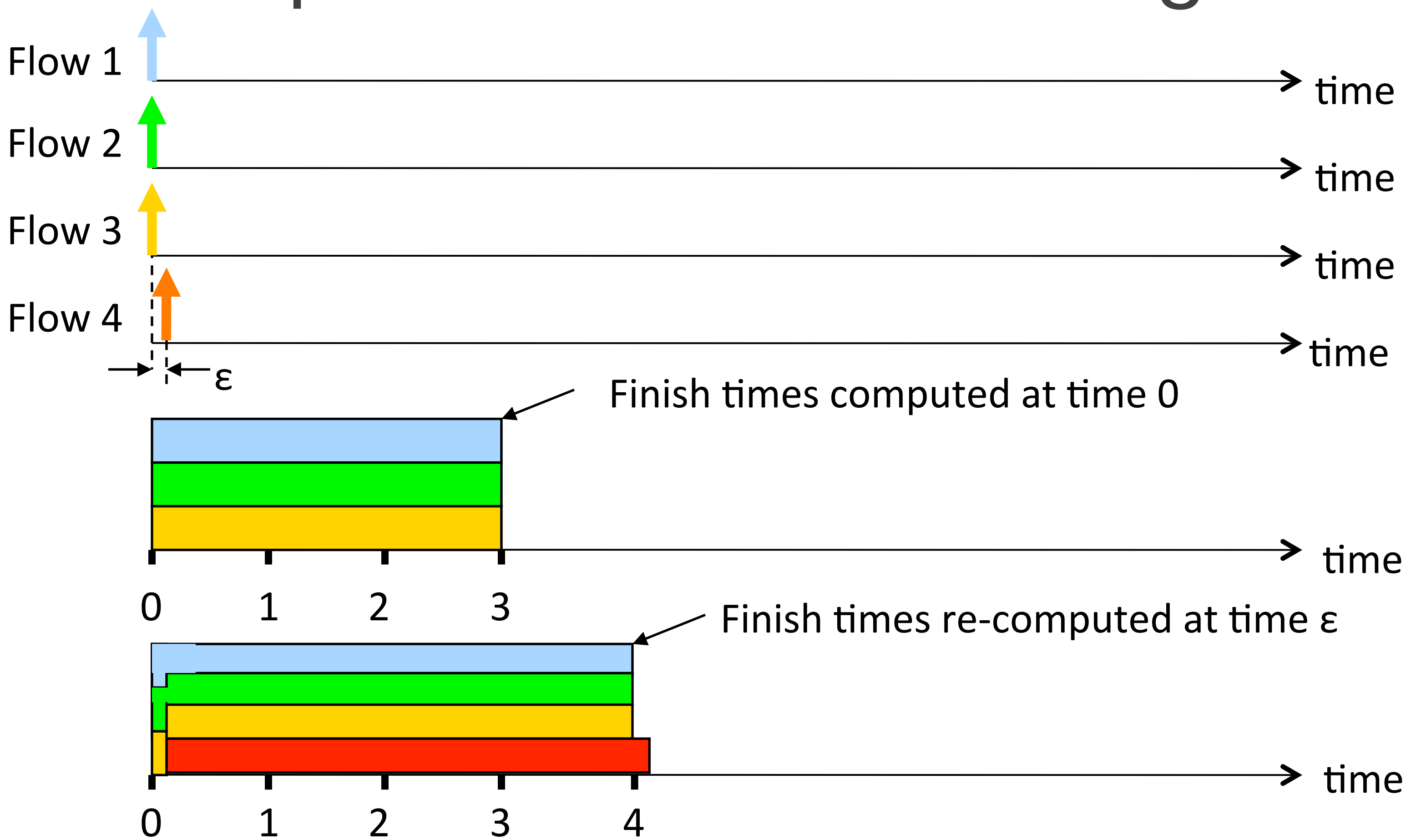
Need to compute the finish time of a packet in the fluid flow system…

… but the finish time may change as new packets arrive!

Need to update the finish times of all packets that are in service in the fluid flow system when a new packet arrives

- But this is very expensive; a high speed router may need to handle hundred of thousands of flows!

Example: Each flow has weight 1



Solution: Virtual Time

Key Observation: while the finish times of packets may change when a new packet arrives, the order in which packets finish doesn't!

- Only the order is important for scheduling

Solution: instead of the packet finish time maintain the number of rounds needed to send the remaining bits of the packet (virtual finishing time)

- Virtual finishing time doesn't change when the packet arrives

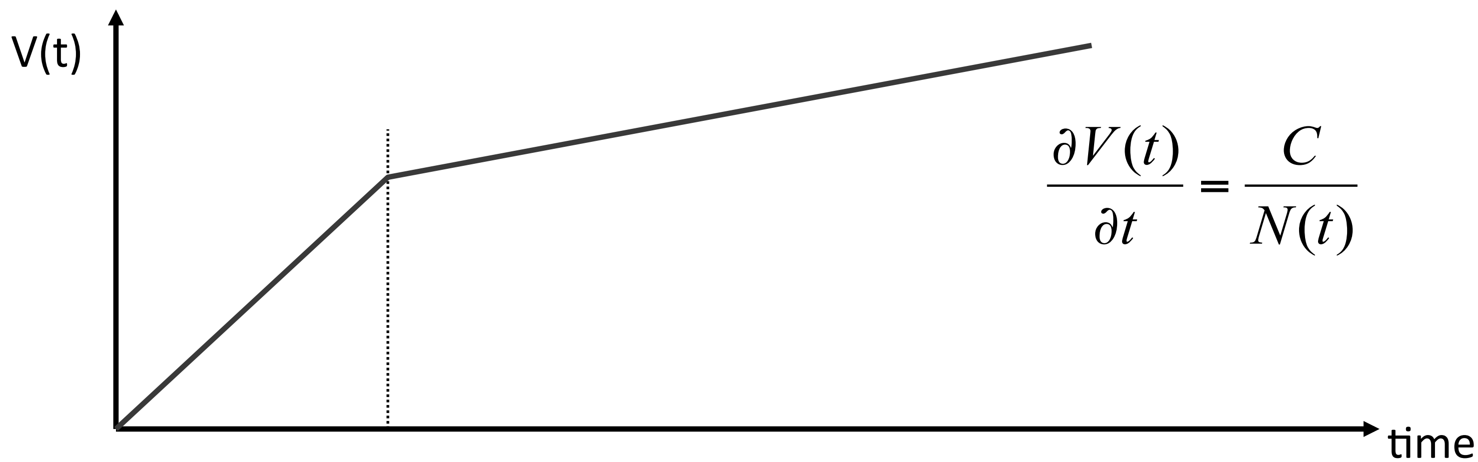
System virtual time – index of the round in the bit-by-bit round robin scheme

System Virtual Time: *V*(t)

Measure service, instead of time

V(t) slope – normalized rate at which every backlogged flow receives service in the fluid flow system

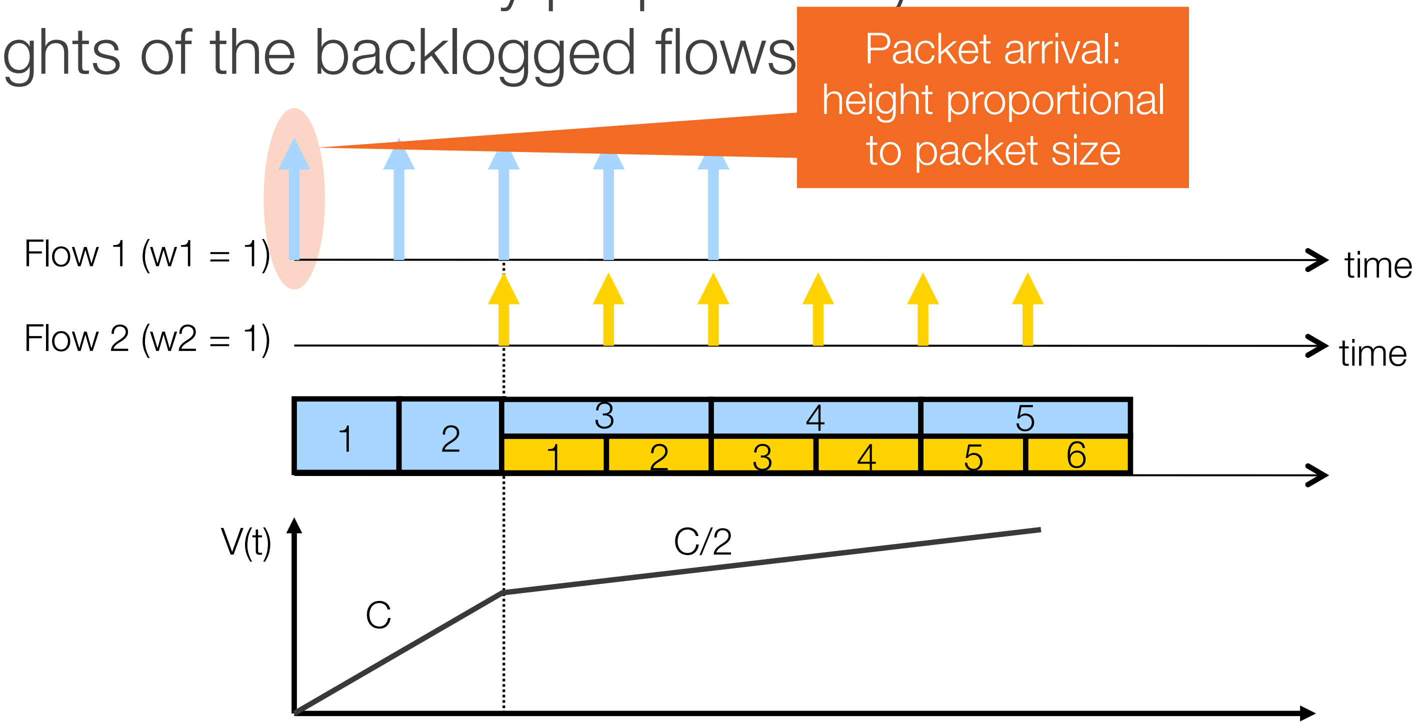
- *C* – link capacity
- *N*(t) – total weight of backlogged flows in fluid flow system at time t



$$\frac{\partial V(t)}{\partial t} = \frac{C}{N(t)}$$

System Virtual Time (V(t)): Example

V(t) increases inversely proportionally to the sum of the weights of the backlogged flows



Fair Queueing Implementation

Define

- F_i^k - virtual finishing time of packet k of flow i
- a_i^k - arrival time of packet k of flow i
- L_i^k - length of packet k of flow i
- w_i – weight of flow i

Current round

Round when last packet of flow *i* finishes

The finishing time of packet $k+1$ of flow i is

# of rounds it takes to serve new packet

$$F_i^{k+1} = \max(V(a_i^{k+1}), F_i^k) + L_i^{k+1} / w_i$$

Round by which each packet is served

Lottery Scheduling

An approximation of weighted fair sharing

- Weight → number of tickets
- Scheduling decision → probabilistic: give a slot to a process proportionally to its weight

Fairness analysis (I)

Lottery scheduling is ***probabilistically fair***

If a thread has a t tickets out of T

- Its probability of winning a lottery is $p = t/T$
- Its expected number of wins over n drawings is np
  - Binomial distribution
  - Variance $\sigma^2 = np(1 - p)$

Fairness analysis (II)

Coefficient of variation of number of wins $\sigma/np = \sqrt{(1-p)/np}$

- Decreases with $\sqrt{n}$

Number of tries before winning the lottery follows a ***geometric distribution***

As time passes, each thread ends receiving its share of the resource

Introduces a bunch of abstractions and mechanisms

Ticket transfers

Ticket currencies

Compensation tickets

Applied to other resources, e.g., memory

Ticket transfers

Explicit transfers of tickets from one client to another

They an be used whenever a client blocks due to some dependency

- When a client waits for a reply from a server, it can temporarily transfer its tickets to the server

They eliminate *priority inversions*

Ticket inflation

Lets users create new tickets

- Like printing their own money
- Counterpart is *ticket deflation*

Normally disallowed except among mutually trusting clients

- Lets them to adjust their priorities dynamically without explicit communication

Ticket currencies (I)

Consider the case of a user managing multiple threads

- Want to let her favor some threads over others
- Without impacting the threads of other users

Ticket currencies (II)

Will let her create new tickets but will debase the individual values of all the tickets she owns

- Her tickets will be expressed in a new *currency* that will have a variable *exchange rate* with the *base currency*

Example (I)

Ann manages three threads

- A has 5 tickets
- B has 3 tickets
- C has 2 tickets

Ann creates 5 extra tickets and assigns them to process C

- Ann now has 15 tickets

Example (II)

These 15 tickets represent 15 units of a new currency whose exchange rate with the base currency is 10/15

The total value of Ann tickets expressed in the base currency is still equal to 10

Compensation tickets (I)

I/O-bound threads are likely get less than their fair share of the CPU because they often block before their CPU quantum expires

Compensation tickets address this imbalance

Compensation tickets (II)

A client that consumes only a fraction f of its CPU quantum ***can*** be granted a ***compensation ticket***

- Ticket inflates the value of all client tickets by 1/f until the client starts gets the CPU
  - *(Wording in the paper is much more abstract)*

Example

CPU quantum is 100 ms

Client A releases the CPU after 20ms

- *f* = 0.2 or 1/5

Value of ***all*** tickets owned by A will be multiplied by 5 until A gets the CPU

Compensation tickets (III)

Compensation tickets

- Favor I/O-bound—and interactive—threads
- Helps them getting their fair share of the CPU

Summary (I)

Weighted max-min fairness a very useful abstraction with strong properties

- Provides isolation (sharing guarantee)
- Strategy proof (cannot be gained)
- Efficient resource usage (if someone doesn’t use resource someone else can use it)
- Can emulate lots of scheduling policies

Summary (I|)

Lottery scheduling

- An approximation of Weighted Fair Queueing in processor domain
- Introduces a bunch of useful abstractions

Dominant Resource Fairness

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

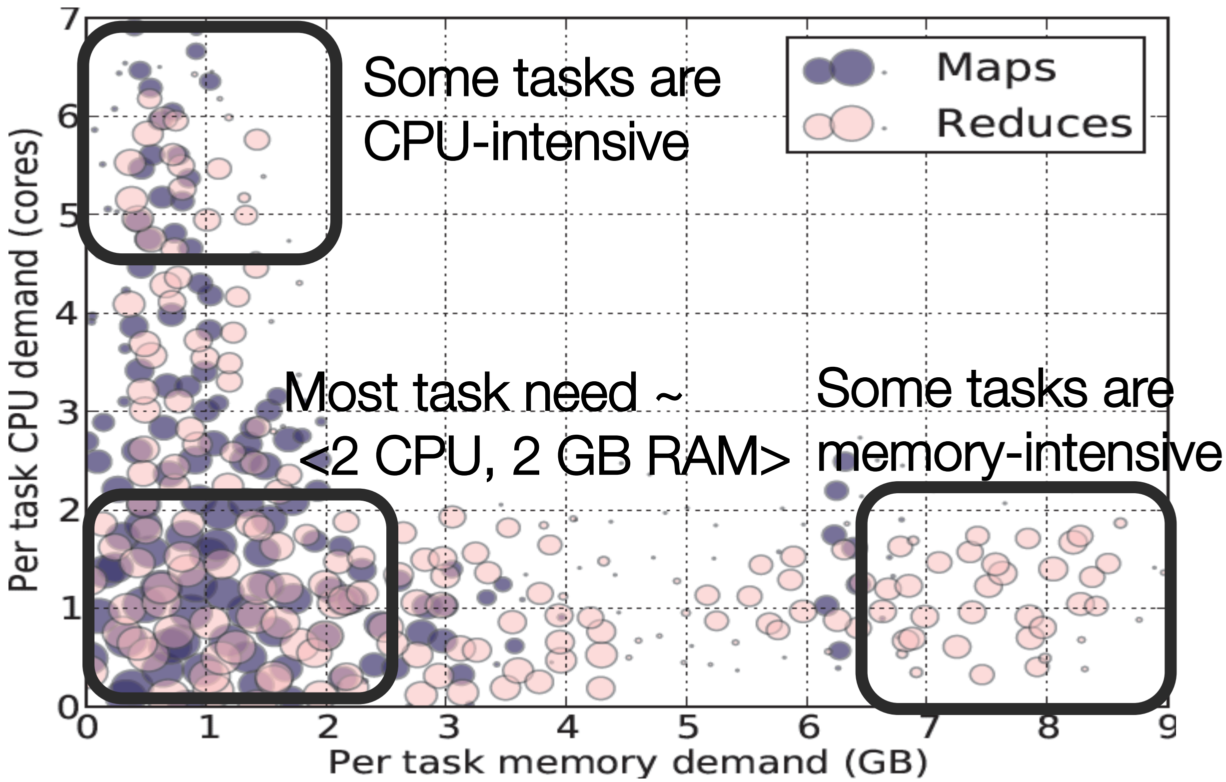
Why is Max-Min Fairness Not Enough?

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)

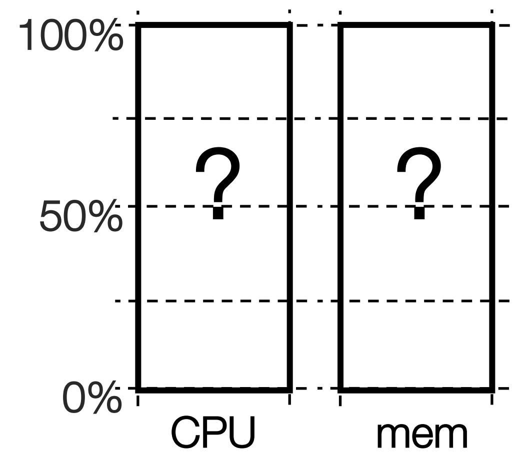
Problem

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***

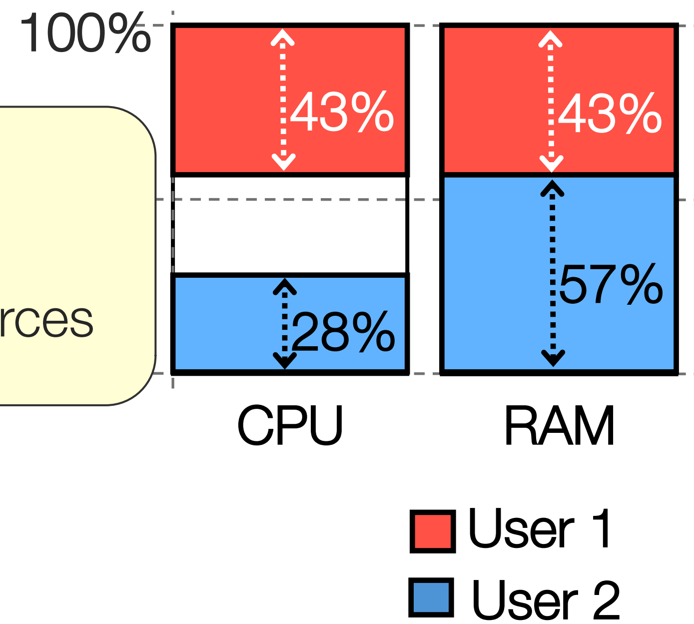
Cluster with 28 CPUs, 56 GB RAM

<u>Problem:</u> 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: $\boxed{\text{<43\% CPUs, 43\% RAM>}}$ (∑=86%)
- U_2: 8 tasks: <28% CPUs, 57% RAM> (∑=86%)



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

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

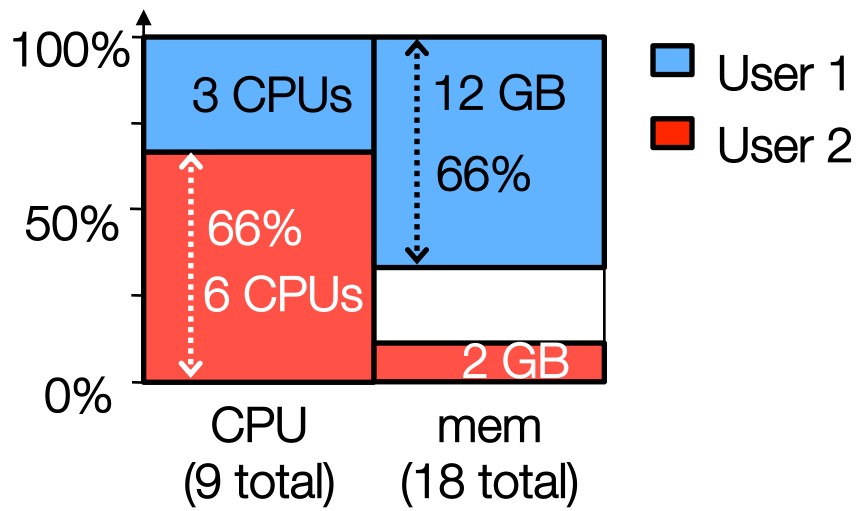
- User 1's dominant share is **25%**

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

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!

Properties of Policies

Property	**Asset**	**CEEI**	**DRF**
Share guarantee		✔	✔
Strategy-proofness	✔		✔
P			
E			
S			
Bottleneck res. fairness		✔	✔
Population monotonicity	✔		✔
Resource monotonicity			

Conjecture: Assuming non-zero demands, DRF is the *only* allocation that is strategy proof and provides sharing incentive (*Eric Friedman, Cornell*)

Summary

Scheduling necessary to deal with oversubscribed resources

- Need to provide isolation
- Need high resource efficiency

Weighted fair queuing achieves many desirable properties

- Many mechanisms to implement it (e.g., Lottery scheduling)
- But limited to a single resource..

Dominant Resource Fairness (DRF):

- Schedules resources of multiple types while preserving WFQ properties