Dominant Resource Fairness

Ali Ghodsi
(presented by Anthony D. Joseph)

LASER Summer School
September 2013
My Talks at LASER 2013

1. AMP Lab introduction
2. The Datacenter Needs an Operating System
3. Mesos, part one
4. Dominant Resource Fairness
5. Mesos, part two
6. Spark
Researchers

- Ali Ghodsi
- Matei Zaharia
- Ben Hindman
- Andy Konwinski
- Scott Shenker
- Ion Stoica

What is Fair Sharing?

n users want to share a resource (e.g., CPU)
  » Solution:
    Allocate each $\frac{1}{n}$ 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 is Fair Sharing Useful?

**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

**Isolation Policy**
- Users cannot affect others beyond their fair share
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 resources 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, MS Quincy
» OS: rr, prop sharing, lottery, linux cfs, …
» Networking: wfq, wf2q, sfq, drr, csfq, …
When is Max-Min Fairness not Enough?

Need to schedule *multiple, heterogeneous* resources

» Example: Task scheduling in datacenters
  - Tasks consume more than just CPU – CPU, memory, disk, and I/O

What are today’s datacenter task demands?
Heterogeneous Resource Demands

Most tasks need ~<2 CPU, 2 GB RAM>

Some tasks are memory-intensive
Some tasks are CPU-intensive

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 & memory
- User 1 wants <1 CPU, 4 GB> per task
- User 2 wants <3 CPU, 1 GB> per task
- What is a fair allocation?
Problem definition

How to fairly share multiple resources when users have heterogeneous demands on them?
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, can simply measure actual consumption

Resources given in multiples of demand vectors

Assume divisible resources
What is Fair?

Goal: define a fair allocation of multiple cluster resources between multiple users

Example: suppose we have:
» 30 CPUs and 30 GB RAM
» Two users with equal shares
» User 1 needs <1 CPU, 1 GB RAM> per task
» User 2 needs <1 CPU, 3 GB RAM> per task

What is a fair allocation?
Definition 1: Asset Fairness

Idea: give weights to resources (e.g. 1 CPU = 1 GB) and equalize total value given to each user

Algorithm: when resources are free, offer to whoever currently has lowest total value
Definition 1: Asset Fairness

Idea: give weights to resources (e.g. 1 CPU = 1 GB) and equalize total value given to each user

Algorithm: when resources are free, offer to whoever currently has lowest total value

Result:

» U1: 12 tasks: 12 CPUs, 12 GB ($24)
» U2: 6 tasks: 6 CPUs, 18 GB ($24)
Idea: give weights to resources (e.g. 1 CPU = 1 GB) and equalize total value given to each user

Algorithm: when resources are free, offer to whoever currently has lowest total value

Result:
- U1: 12 tasks: 12 CPUs, 12 GB ($24)
- U2: 6 tasks: 6 CPUs, 18 GB ($24)

**PROBLEM**
User 1 has < 50% of both CPUs and RAM

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

**Definition 1: Asset Fairness**

<table>
<thead>
<tr>
<th>CPU</th>
<th>User 1</th>
<th>User 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td>RAM</td>
<td>30 CPUs / 30 GB RAM</td>
<td>30 CPUs / 30 GB RAM</td>
</tr>
</tbody>
</table>

Result:
- » U1: 12 tasks: **12 CPUs, 12 GB** ($24)
- » U2: 6 tasks: 6 CPUs, 18 GB ($24)
Lessons from Definition 1

“You shouldn’t do worse than if you ran a smaller, private cluster equal in size to your fair share”

Thus, given $N$ users, each user should get $\geq 1/N$ of her *dominating* resource (i.e., the resource that she consumes most of)
Cheating the Scheduler

Some users will *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
Two Important Properties

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

**Envy-freeness**
- No user would ever strictly prefer another user’s lot in an allocation
- **Intuition:**
  - Don’t want to trade places with any other user
Challenge

A fair sharing policy that provides
» Strategy-proofness
» Share guarantee

Max-min fairness for a single resource had these properties
» Generalize max-min fairness to multiple resources
Dominant Resource Fairness

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

» Example:

  Total resources: \(<10 \text{ CPU}, \ 4 \text{ GB}>\)
  User 1’s allocation: \(<2 \text{ CPU}, \ 1 \text{ GB}>\)
  Dominant resource is memory as \(\frac{1}{4} > \frac{2}{10} (1/5)\)

A user’s *dominant share* is the fraction of the dominant resource she is allocated

» User 1’s dominant share is 25% (1/4)
Def. 2: Dominant Resource Fairness

Idea: Apply max-min fairness to dominant shares
- Give every user an equal share of her dominant resource (i.e., resource it consumes most of)

Algorithm: when resources are free, offer to the user with the smallest dominant share (i.e., fractional share of her dominant resource)

Result:
- U1: 15 tasks: 15 CPUs, 15 GB
- U2: 5 tasks: 5 CPUs, 15 GB

![Diagram showing CPU and RAM usage for User 1 and User 2]
DRF is Fair

DRF is strategy-proof
DRF satisfies the share guarantee
DRF allocations are envy-free

See DRF NSDI’11 paper for proofs
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

Need to determine demand vectors
Determining 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
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

- Dominant resource is memory: <1 CPU, 10 GB>
- <1 CPU, 1 GB>
- Dominant resource is CPU:
- <2 CPU, 4 GB>
- <1 CPU, 3 GB>
- Dominant Shares:
- Share guarantee: ~50% dominant share
- Share guarantee: ~70% dominant share

User 1's Shares

User 2's Shares

Dominant resource is CPU
How is Fairness Solved in Datacenters Today?

Hadoop Fair Scheduler, capacity-based schedulers, Microsoft’s 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

Ten minute experiment
- 48-node Mesos EC2 cluster with 8 CPU cores 6 GB
- DRF versus 3 – 6 slots per node
- Oversubscription of machine resources allowed for slots

Two entities
- Four users submitting small jobs <1 CPU, 0.5GB>
- Four users submitting large jobs <2 CPU, 2GB>
- Each job consisted of 80 tasks
Experiment: DRF vs Slots

Number of Type 1 Jobs Finished in 10 minutes

- DRF: 35 jobs
- 3 slots: 33 jobs
- 4 slots: 30 jobs
- 5 slots: 17 jobs
- 6 slots: 8 jobs

Thrashing

Number of Type 2 Jobs Finished in 10 minutes

- DRF: 91 jobs
- 3 slots: 37 jobs
- 4 slots: 61 jobs
- 5 slots: 66 jobs
- 6 slots: 35 jobs

Low utilization

Thrashing

Type 1 jobs: <2 CPU, 2 GB>
Type 2 jobs: <1 CPU, 0.5GB>
Experiment: DRF vs Slots

Completion Time of Type 1 Jobs

<table>
<thead>
<tr>
<th>Type 1 job: &lt;2 CPU, 2 GB&gt;</th>
<th>Completion Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRF</td>
<td>65</td>
</tr>
<tr>
<td>3 slots</td>
<td>69</td>
</tr>
<tr>
<td>4 slots</td>
<td>72</td>
</tr>
<tr>
<td>5 slots</td>
<td>123</td>
</tr>
<tr>
<td>6 slots</td>
<td>196</td>
</tr>
</tbody>
</table>

Thrashing

Completion Time of Type 2 Jobs

<table>
<thead>
<tr>
<th>Type 2 job: &lt;1 CPU, 0.5 GB&gt;</th>
<th>Completion Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRF</td>
<td>25</td>
</tr>
<tr>
<td>3 slots</td>
<td>61</td>
</tr>
<tr>
<td>4 slots</td>
<td>39</td>
</tr>
<tr>
<td>5 slots</td>
<td>35</td>
</tr>
<tr>
<td>6 slots</td>
<td>56</td>
</tr>
</tbody>
</table>

Low utilization hurts performance

Thrashing
Reduction in Job Completion Time
DRF vs Slots

Simulation of 1-week Facebook traces

Completion Time Reduction

Job Size (tasks)

-3%  35%  51%  48%  55%  66%  53%
1-500 501-1000 1001-1500 1501-2000 2501-3000 2501-3000 3001-∞
Reduction in Job Completion Time

DRF vs Slots

Simulation of 1-week Facebook traces

Completion time dominated by longest task

Multiple execution phases benefit from higher utilization
Utilization of DRF vs Slots

Simulation of Facebook workload

CPU Utilization

Memory Utilization

Time (s)
Summary

DRF provides *multiple-resource fairness* in the presence of *heterogeneous 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
My Talks at LASER 2013

1. AMP Lab introduction
2. The Datacenter Needs an Operating System
3. Mesos, part one
4. Dominant Resource Fairness
5. Mesos, part two
6. Spark
DRF Alternative: Use an Economic Model

Approach
  » Set prices for each good
  » Let users buy what they want

How do we determine the right prices for different goods?

Let the market determine the prices
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
- User 1 lies and demands <2 CPU, 2 GB>
- Utilization-Optimal scheduler prefers user 1

<table>
<thead>
<tr>
<th>User 1</th>
<th>User 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: 50%</td>
<td>CPU: 95%</td>
</tr>
<tr>
<td>mem: 50%</td>
<td>mem: 50%</td>
</tr>
</tbody>
</table>

- User 1
- User 2

- User 1
- User 2

- User 1
- User 2
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>

a) DRF
b) Asset Fairness
c) CEEI
Max/Min Theorem for DRF

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.
Desirable Fairness Properties (1)

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

*Progressive filling (PF) algorithm*

1. Allocate \( \varepsilon \) to every flow until some link saturated
2. Freeze allocation of all flows on saturated link and goto 1
Desirable Fairness Properties (2)

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
Desirable Fairness Properties (3)

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

DRF will allocate same dominant share to all users
  » As soon as PF saturates a resource, allocation frozen
Desirable Fairness Properties (4)

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

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
<table>
<thead>
<tr>
<th>Property</th>
<th>Asset</th>
<th>CEEI</th>
<th>DRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share guarantee</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Strategy-proofness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Pareto efficiency</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Envy-freeness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Single resource fairness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Bottleneck res. fairness</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Population monotonicity</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Resource monotonicity</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Demands at Facebook

CDF of tasks vs. Ratio of task demand to resource per slot

- Green line: Memory demand
- Blue line: CPU demand