H-DRF: Hierarchical Scheduling for Diverse Datacenter Workloads

Ali Ghodsi
join work with
Arka Bhattacharya, David Culler, Eric Friedman, Ion Stoica, Scott Shenker

UC Berkeley
Data centers run a large mix of workloads...leading to diverse resource requirements.
multi-resource scheduling necessary for isolation and efficiency
Background: multi-resource fairness

• **Dominant Resource Fairness (DRF)**
  – Share guarantee: guaranteed 1/n share
  – Strategy-proof: lying can only hurt you

• Well understood
  – Efficiency, extensions, limitations

• DRF now de-facto scheduler in Hadoop
  – DRF capacity scheduler (HortonWorks)
  – DRF fair scheduler (Cloudera)
Slight problem…

• Hadoop always had *hierarchical policies*
  – Problem: DRF didn’t mention hierarchies

• Both industry implementations adapted DRF to support hierarchies
What’s hierarchical scheduling?
Hierarchical Scheduling

- Entire Cluster
  - Ads (60%)
  - QA (50% of Dev)
  - Test (50% of Dev)
  - Analytics (100% of ads)

- Dev. (40%)
Hierarchical Scheduling

Entire Cluster

Ads (60%)

Analytics
(100% of ads)

Dev. (40%)

QA
(100% of Dev)
Multi-Resource Scheduling + Hierarchical Policies = Challenging

• Hadoop DRF schedulers can break down
  – Leave resources unallocated, or
  – Starve some users
Problem Statement

How to generalize DRF to support hierarchical policies?

Dominant Resource Fairness + Hierarchical Scheduling
Problem Statement

How to generalize DRF to support hierarchical policies?

Dominant Resource Fairness

- Share guarantee
- $1/n$ share to leaves
- Pareto efficiency
- Work-conservation

Hierarchical Scheduling

- Hierarchical share guarantee
- $1/n$ to every node
- Pareto efficiency
- Work-conservation
Outline

• How to schedule multi-resources? (DRF)
• Why is it challenging?
• What’s our solution? (H-DRF)
• How well does it work?
Dominant Resource Fairness (DRF)

- **Dominant resource** of a user is the resource she has biggest share of
  - **Dominant share** of a user is her share of her dominant resource

Total resources: \(<100 \text{ Cpus, 100 Gpus}>\) (2 types of resources)
User 1 demand: \(<3 \text{ Cpus, 2 Gpus}>\) dom res: \textbf{Cpu}
User 2 demand: \(<2 \text{ Cpus, 3 Gpus}>\) dom res: \textbf{Gpu}

- **DRF Scheduler**
  - Max-min fair allocation on dominant shares
  - "Equalize" the dominant share of all users
Outline

• How to schedule multi-resources? (DRF)
• Why is it challenging?
• What’s our solution? (H-DRF)
• How well does it work?
Hierarchy Flattening

• General technique
  – Compute fair share of every leaf node
  – Use **weighted** scheduler (weighted DRF)

• Works for **any** single-resource scheduler
Hierarchy Flattening

- Ads
  - Share guarantee: 50% of cluster
    - Anlt
      - Share Guarantee: 50% of cluster
    - Test
      - Share Guarantee: 25% of cluster
  - Share Guarantee: 50% of cluster
- Dev
  - Share Guarantee: 50% of cluster
    - QA
      - Share Guarantee: 25% of cluster
Hierarchy Flattening

- Ads
  - Anlt: Weight: 2
- Dev
  - Test: Weight: 1
  - QA: Weight: 1

Share guarantee: 50% of cluster
Share Guarantee: 50% of cluster
Total resources: <100 Cpus, 100 Gpus>

Anlt
Weight: 2
Demand: <1, 1>

Test
Weight: 1
Demand: <1, 0>

QA
Weight: 1
Demand: <0, 1>
Initial Allocation

<table>
<thead>
<tr>
<th>Resource 1</th>
<th>Resource 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anlt: 2 units</td>
<td>Anlt: 2 units</td>
</tr>
<tr>
<td>Test: 1 unit</td>
<td>QA: 1 unit</td>
</tr>
</tbody>
</table>

Demand: <1, 1>  
Demand: <1, 0>  
Demand: <0, 1>

Weight: 2  
Weight: 1  
Weight: 1
Final Allocation

<table>
<thead>
<tr>
<th>Resource 1</th>
<th>Resource 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test: 33%</td>
<td>QA: 33%</td>
</tr>
<tr>
<td>Anlt: 66%</td>
<td>Anlt: 66%</td>
</tr>
</tbody>
</table>

Demand: <1, 1>  
Weight: 2

Test  
Demand: <1,0>  
Weight: 1

QA  
Demand: <0,1>  
Weight: 1
Hierarchical Share Guarantee Violated

Share Guarantees:

- Ads: 50%
- Anlt: 50%
- Dev: 50%
- Test: 25%
- QA: 25%

Final Allocation:

- Resource 1:
  - Anlt: 66%
  - Test: 33%
- Resource 2:
  - Anlt: 66%
  - QA: 33%

Dev: 33%
Outline

• How to schedule multi-resources? (DRF)
• Why is it challenging?
• What’s our solution? (H-DRF)
• How well does it work?
Static H-DRF

- Traverse tree top to bottom
  - Recursively pick node with smallest dom. share
  - Top-down equalize siblings
Total Resources: <100,100>

- Ads
  - Anlt <1,0>
  - Dev
    - Test <1,0>
    - QA <0,1>
Ideal DRF allocations in hierarchy

Hierarchical share guarantees for every node
Starvation

Allocation in a dynamic cluster

Resource 1

Anlt 100%

QA 100%

Resource 2

Ideal allocation

Resource 1

Test 50%

Anlt 50%

QA 100%

Resource 2
Outline

• How to schedule multi-resources? (DRF)
• Why is it challenging?
• What’s our solution? (H-DRF)
• How well does it work?
Hierarchical DRF (H-DRF)

• Leverage Static H-DRF

• Add two invariants
  – Re-scale consumption vectors
  – Ignore terminated/blocked nodes
Re-scaling Consumption Vectors

• **Intuition**
  – No starvation from empty cluster
  – Rescale back as if started from empty cluster

• **Re-scaling**
  – Choose sibling with lowest dominant share $M$
  – Rescale all siblings to have a dominant share $M$
  – Parent resource usage = sum of rescaled vectors
Example

Min. dom. share: **Test (49)**

Rescale siblings: **QA <0, 49>**

Dev’s vector: <49,0>+<0, 49> = <49,49>
dom share : 50%

- Ads
  - Anlt
    - Test
      - QA

dom share : 49%

- Dev
  - Ads
    - Anlt
      - Test
        - QA
Hierarchical DRF (H-DRF)

• Leverage Static H-DRF

• Add two invariants
  – Re-scale consumption vectors
  – Ignore terminated/blocked nodes

---

**Static H-DRF**

• Traverse tree top to bottom
  – Recursively pick node with smallest dom. Share
  – Equalize siblings

---

**Algorithm**

```plaintext
R = (r_1, \ldots, r_m) \triangleright total resource capacities
C = (c_1, \ldots, c_m) \triangleright current consumed resources
W \triangleright resources to allocate
A \triangleright set of nonzero resources in W
Y \triangleright set of nonleaf nodes in Y or parents of demanding nodes
n_r \triangleright root node in hierarchy tree
C(n) \triangleright children of any node n
s_i (i = 1 \ldots n) \triangleright dominant shares
U_i = (u_{i,1}, \ldots, u_{i,m}) (i = 1 \ldots n) \triangleright "scaled" resources

function (recursive) UpdateS(n_i)
if n_i is a leaf node then
  s_i = max U_{i,j} / R_j for j \in Y
  return U_i
else
  Q = set of U_j's from UpdateS(n_j) for children of n_i
  f = maximum dominant share from Q restricting to nodes in A and resources in Y
  Rescale demanding vectors in Q by f
  U_i = sum of vectors in Q
  s_i = max U_{i,j} / R_j for j \in Y
  return U_i

function Alloc(W)
  n_i = n_r
  while n_i is not a leaf node (job) do
    n_j = node with lowest dominant share s_j in C(n_i), which also has a task in its subtree that can be scheduled using W
    n_i = n_j
    D_i = max \{ W_{i,j} T_j \} \text{ s.t. } T_i is n_i's task demand vector
    C = C + D_i \triangleright update consumed vector
    U_i = U_i + D_i \triangleright update leaf only
    Recompute s: UpdateS(n_i)
    Allocate the resources: Alloc(W)
```
Example

‘Dev’ keeps getting selected because it has 0% dom share

QA has no tasks
Ignore **Blocked** Nodes

**• A node is blocked iff**

- No more demand
- Cannot be allocated more resources
- All its children are blocked

**• Ignore blocked nodes**

- Only look at non-blocked siblings for min $M$
- Rescale non-blocked nodes to dominant share $M$
Ignoring terminated/ blocked nodes

- **Dev** node with the blocking status of 49/0 (r1/r2)
- **Test** node with the status of <1, 0>
- **QA** node with the status of <0, 1>

The diagram shows the distribution of tasks between the nodes.
Outline

• How to schedule multi-resources? (DRF)
• Why is it challenging?
• What’s our solution? (H-DRF)
• How well does it work?
Evaluation

• 50 EC2 nodes having 6 GB memory, 4 CPUs and 1 GPU each.

• Evaluated against
  – Hadoop Capacity Scheduler (not Pareto)
  – Hadoop Capacity Scheduler (Pareto added)

• Input: A 100-job schedule containing a mix of large and small jobs
Hierarchy Used

Mixed

Small Jobs

Large Jobs

$N_{mixed}$

$N_{small}$

$N_{small_{1}}$, $N_{small_{2}}$, $N_{small_{3}}$

$N_{large_{1}}$

$N_{CPU}$, $N_{GPU}$
Throughput

 Median Throughput (# Tasks)

\[
\begin{align*}
N_{\text{CPU}} & : 83, 79 \\
N_{\text{GPU}} & : 49, 21 \\
N_{\text{small,1}} & : 9, 9 \\
N_{\text{small,2}} & : 14, 14 \\
N_{\text{small,3}} & : 8, 7 \\
N_{\text{large,1}} & : 84, 77
\end{align*}
\]

- **HDRF**
- **C.S-Current**

Pareto violated
Throughput

Starvation

Leaf Nodes

Median Throughput (# Tasks)

Pareto-Efficient C.S

HDRF

N_{CPU} 83

N_{GPU} 48 49

N_{small,1} 6 9

N_{small,2} 11 14

N_{small,3} 6 8

N_{large,1} 158 84
Conclusion

• Hierarchical scheduling policies important

• Hierarchical + Multi-resource = Challenging
  – Starvation, or violation of share guarantees

• Proposed \textit{H-DRF}
  – \textit{Generalization} of DRF to hierarchies
  – Guards against starvation
  – Provides hierarchical share guarantee
Thank you
**Algorithm**

\[
R = \langle r_1, \ldots, r_m \rangle \quad \triangleright \text{total resource capacities}
\]

\[
C = \langle c_1, \ldots, c_m \rangle \quad \triangleright \text{current consumed resources}
\]

\[W \text{ resources to allocate} \quad \triangleright \text{Assumption: } R - C > W\]

\[Y \text{ set of nonzero resources in } W\]

\[A \text{ (demanding), set of leaf nodes that use only resources in } Y \text{ or parents of demanding nodes}\]

\[n_r \quad \triangleright \text{root node in hierarchy tree}\]

\[C(n) \quad \triangleright \text{children of any node } n\]

\[s_i \ (i = 1 \ldots n) \quad \triangleright \text{dominant shares}\]

\[U_i = \langle u_{i,1}, \ldots, u_{i,m} \rangle \ (i = 1 \ldots n) \quad \triangleright \text{“scaled” resources}\]

**Recompute s:** \(UpdateS(n_r)\)

**Allocate the resources:** \(Alloc(W)\)

---

**function (recursive) UpdateS(n_i)**

if \(n_i\) is a leaf node then

\[s_i = \max U_{ij} / R_j \text{ for } j \in Y\]

return \(U_i\)

else

\[Q = \text{set of } U_j\’s \text{ from } UpdateS(n_j) \text{ for children of } n_i\]

\[f = \text{maximum dominant share from } Q \text{ restricting to nodes in } A \text{ and resources in } Y\]

Rescale demanding vectors in \(Q\) by \(f\)

\[U_i = \text{sum of vectors in } Q\]

\[s_i = \max U_{i,j} / R_j \text{ for } j \in Y\]

return \(U_i\)

---

**function Alloc(W)**

\[n_i = n_r\]

while \(n_i\) is not a leaf node (job) do

\[n_j = \text{node with lowest dominant share } s_j \text{ in } C(n_i)\], which also has a task in its subtree that can be scheduled using \(W\)

\[n_i = n_j\]

\[D_i = \frac{W_i}{\max_j\{T_{i,j}\}} T_i\], s.t. \(T_i\) is \(n_i\’s\ task demand vector\]

\[C = C + D_i \quad \triangleright \text{update consumed vector}\]

\[U_i = U_i + D_i \quad \triangleright \text{update leaf only}\]