Tetris
Multi-Resource Packing for Cluster Schedulers

Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, Aditya Akella
We find that:

- Resources are *fragmented* i.e. machines run below capacity
- Even at 100% usage, goodput is smaller due to *over-allocation*
- Pareto-efficient multi-resource fair schemes do not lead to good avg. performance

**Tetris**

*Up to 40% improvement in makespan*¹ *and job completion time with near-perfect fairness*

¹Time to finish a set of jobs
Findings from Bing and Facebook traces analysis

Applications have (very) diverse resource needs

- Tasks need *varying amounts of each resource*
- *Demands* for resources are weakly correlated

This matters, because no single bottleneck resource in the cluster:

- E.g., enough cross-rack network bandwidth to use all cores

**Upper bound on potential gains**

- *Makespan reduces by \(\approx 49\%\)*
- *Avg. job completion time reduces by \(\approx 46\%\)*
Production schedulers *neither pack tasks nor consider all their relevant resource demands*

#1 Resource Fragmentation

#2 Over-allocation
Resource Fragmentation (RF)

Current Schedulers

“Packer” Scheduler

Machine A
4 GB Memory

Machine B
4 GB Memory

T1: 2 GB

T2: 2 GB

T3: 4 GB

Avg. task compl. time = 1.33 t

Avg. task compl. time = 1 t

Allocate resources per slots, fairness.

Are not explicit about packing.

RF increase with the number of resources being allocated!
Over-Allocation

Not all of the resources are explicitly allocated.

E.g., disk and network can be over-allocated.
Why so bad #2
Multi-resource Fairness Schemes do not solve the problem

Example in paper
Packer vs. DRF: makespan and avg. completion time improve by over 30%

Work Conserving ≠ no fragmentation, over-allocation

Pareto\(^1\) efficient ≠ performant
- Treat cluster as a big bag of resources
  - Hides the impact of resource fragmentation
- Assume job has a fixed resource profile
  - Different tasks in the same job have different demands
  - How the job is scheduled impacts jobs’ current resource profiles
  - Can schedule to create complementarity

\(^1\)no job can increase its share without decreasing the share of another
Current Schedulers

1. Resource Fragmentation
2. Over-Allocation
3. Fair allocations sacrifice performance

Competing objectives

Cluster efficiency vs. Job completion time vs. Fairness
Pack tasks along multiple resources to improve cluster efficiency and reduce makespan
**Theory**

Multi-Resource Packing of Tasks
similar to
Multi-Dimensional Bin Packing

Avoiding fragmentation looks like:
- Tight bin packing
- Reduce # of bins → **reduce makespan**

**Practice**

Existing heuristics do not directly apply:
- Assume balls of a fixed size
- Assume balls are known apriori

- vary with time / machine placed
- elastic
- cope with online arrival of jobs, dependencies, cluster activity

---

1APX-Hard is a strict subset of NP-hard
Balls could be tasks
Bin could be machine, time
A packing heuristic

- Tasks resources demand vector
- Machine resource vector

Alignment score (A)

“A” works because:

1. Check for fit to ensure no over-allocation

2. Bigger balls get bigger scores

3. Abundant resources used first
# 2

Faster average job completion time
Q: What is the shortest “remaining time”? 

“remaining work” = \[ \text{remaining # tasks} \& \text{tasks’ resource demands} \& \text{tasks’ durations} \]

A job completion time heuristic

- Gives a score \( P \) to every job
- Extended SRTF to incorporate multiple resources

Shortest Remaining Time First\(^1\) (SRTF)

\[ \text{schedules jobs in ascending order of their remaining time} \]

\(^1\)SRTF – M. Harchol-Balter et al. Connection Scheduling in Web Servers [USITS'99]
Combine A and P scores!

1: among J runnable jobs
2: \( \text{score} (j) = A(t, R) + \varepsilon P(j) \)
3: \( \max \text{ task } t \text{ in } j, \text{ demand}(t) \leq R \text{ (resources free)} \)
4: \( \text{pick } j^*, t^* = \arg \max \text{ score}(j) \)
# 3

Achieve performance and fairness
Performance and fairness do not mix well in general  

But ....  
We can get “perfect fairness” and much better performance  

- **Packer** says: “*task T should go next to improve packing efficiency*”  
- **SRTF** says: “schedule *job J to improve avg. completion time*”  
- **Fairness** says: “*this set of jobs should be scheduled next*”  

Possible to satisfy all three  
In fact, happens often in practice
Fairness Knob, $F \in [0, 1)$

- Pick the best-for-perf. task from among $(1-F)$ fraction of jobs furthest from fair share

**Heuristic**

- Fairness Knob, $F \in [0, 1)$

**Fairness is not a tight constraint**

- Lose a bit of fairness for a lot of gains in performance
- Long term fairness not short term fairness

- **Heuristic**
  - $F = 0$ → Most unfair
  - $F \to 1$ → Close to perfect fairness

- **Pick the best-for-perf. task from among $(1-F)$ fraction of jobs furthest from fair share**
We saw:
- Packing efficiency
- Prefer small remaining work
- Fairness knob

Other things in the paper:
- Estimate task demands
- Deal with inaccuracies, barriers
- Other cluster activities

Putting it all together

Job Manager
Multi-resource asks; barrier hint

Node Manager
Track resource usage; enforce allocations

Cluster-wide Resource Manager
New logic to match tasks to machines (+packing, +SRTF, +fairness)

Yarn architecture
Changes to add Tetris (shown in orange)
Evaluation

- Implemented in Yarn 2.4
- 250 machine cluster deployment
- Bing and Facebook workload
Efficiency

Tetris vs. Single Resource Scheduler

<table>
<thead>
<tr>
<th></th>
<th>Makespan</th>
<th>Avg. Job Compl. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Resource Scheduler</td>
<td>29 %</td>
<td>30 %</td>
</tr>
<tr>
<td>Multi-resource Scheduler</td>
<td>28 %</td>
<td>35 %</td>
</tr>
</tbody>
</table>

Gains from
- avoiding fragmentation
- avoiding over-allocation

Low value → high fragmentation
**Fairness Knob**

- quantifies the extent to which Tetris adheres to fair allocation

---

<table>
<thead>
<tr>
<th></th>
<th>Makespan</th>
<th>Job Compl. Time</th>
<th>Avg. Slowdown [over impacted jobs]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>No Fairness</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F = 0)</td>
<td>50 %</td>
<td>40 %</td>
<td>25 %</td>
</tr>
<tr>
<td><strong>Full Fairness</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(F \rightarrow 1)</td>
<td>10 %</td>
<td>23 %</td>
<td>2 %</td>
</tr>
<tr>
<td>(F = 0.25)</td>
<td>25 %</td>
<td>35 %</td>
<td>5 %</td>
</tr>
</tbody>
</table>
Pack efficiently along multiple resources

Prefer jobs with less "remaining work"

Incorporate Fairness

- Combine heuristics that improve packing efficiency with those that lower average job completion time
- Achieving desired amounts of fairness can coexist with improving cluster performance
- Implemented inside YARN; deployment and trace-driven simulations show encouraging initial results

We are working towards a Yarn check-in