Multi-Resource Packing for Cluster Schedulers

CS6453: Johan Björck

The problem

Tasks in modern cluster environments have a diverse set of resource requirements

CPU, memory, disk, network...

The problem

Current scheduling strategies results in fragmentation and over-allocation of resources that aren’t explicitly allocated (for example bandwidth)

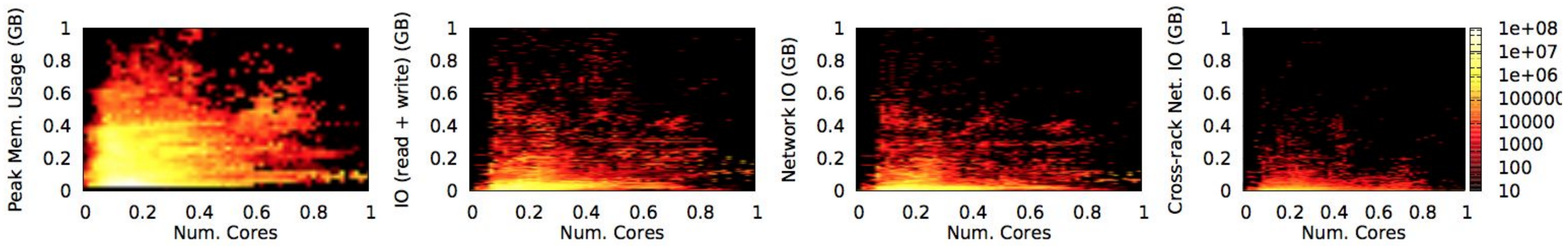
They also focus on fairness to the degree that it might impair efficiency

This leads to poor performance...

Why is this interesting?

Empirically there is a large diversity in resource consumption among tasks, and optimal packing is challenging yet potentially very beneficial

The authors argue that the benefits in makespan could be up to 40%



Limitations of other frameworks

Slots - many frameworks pack jobs based on memory slots, and fixed size slots naturally leads to waste

Just using one metric can also lead to implicit over-allocation, consider for example a reduce task in MapReduce, it has high bandwidth requirement

Limitations of other frameworks

It is also common for scheduling frameworks to allocate resources fairly, for example by giving resources to the job that is the furthest away from a fair allocation

Just focusing on fairness has a potentially large footprint on efficiency, and it might better to balance them

Difference to prior work

Many cluster schedulers focus on data locality or fairness, the article balances these

While there are been much investigation into balancing fairness and efficiency the authors argue that their contribution lies in the implementation and evaluation

Naturally there has been a lot of work in bin packing

Contribution 1 - problem formulation

By looking into the actual data you can empirically find that there is a lot of diversity in resource consumption among tasks

A central contribution is formulating this problem in an intuitive algorithmic framework, and sketching out the potential gains

Contribution 2 - the system

The authors also defines a method for solving the bin packing problem online, while taking practical matters such as fairness and barriers in considerations

The method is also implemented in practice, with positive results

The mathematical mode

The resource allocation problem is modelled as a bin packing problem - we want to fit a number of jobs with specific resource requirements into as few machines as possible, this is already APX-hard

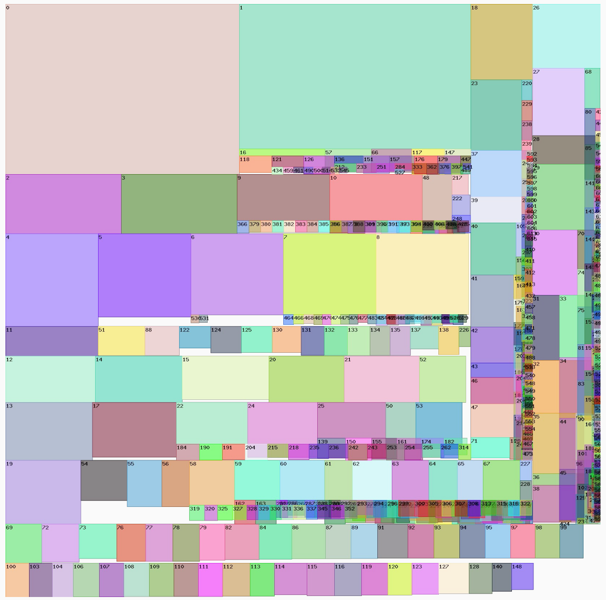
There are many practical differences, the resource requirements are elastic and dynamic, so the problem is even harder

The authors argue that this can bee seen as a hardness result...

Bin packing

A heuristic for the packing problem is given

When a machine is available, for each job that can fit, calculate score as cosine similarity between “vector” of local resources of machine and job



Reducing completion time

To reduce average job completion time, a common method is shortest remaining time first (SRTF), leads to resource fragmentation

This is extended, score a task by its estimated duration time its resource usage, and score a job by the sum of scores for its tasks

We combine this score with the alignment score by a weighted sum, and pick the job with the highest score, with tunable weight

Adding fairness and DAG awareness

To add fairness introduce parameter f, once a machine is free only consider the (1-f) fraction of jobs that have current resource allocation furthest away from fair allocation

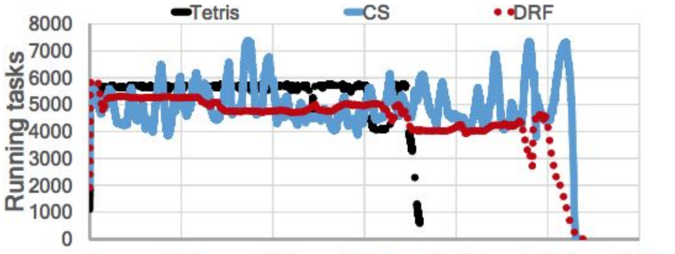
We want to preferentially schedule last few task before a barrier, introduce a parameter b, schedule tasks remaining after a b fraction before a barrier are done preferentially

Implementation

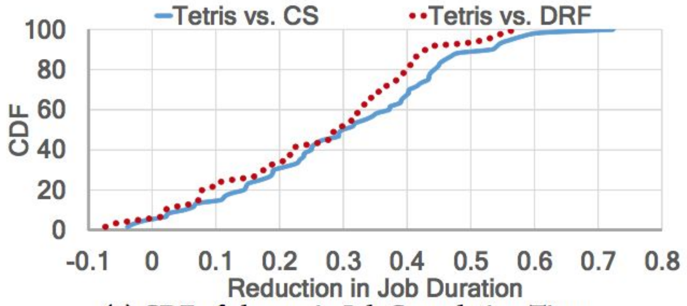
The system is implemented into the Yarn framework, where task to machine matching is done according to earlier sections

Task demand is estimated from historical jobs, allocations are explicitly enforced with a token system, nodes estimate dynamic resource availability

Results



(a) Number of running tasks



(a) CDF of change in Job Completion Time

	Avg.	Stdev.
Tetris vs. CS	29%	2.94%
Tetris vs. DRF	27.5%	3.50%

(b) Makespan Reduction

Figure 4: Comparing Tetris vs. **baselines. The results are from running a workload of over 100 jobs on a 250 server cluster running Yarn 2.4. Each experiment run takes > 10 hours; repeat five times.**

Problem 1: There’s no formal proof

The authors show that the overhead is negligible, however it is harder to show that the method does perform well in general

Potentially the method offers little benefit for additional cost, an example would a multitude of small and predictable jobs where a naive algorithm would perform well without overhead

There’s no proof of general applicability, and the work solely relies on empirical observation of limited sample size

Problem 2: Unpredictable jobs

The system relies on predicting the running time for jobs, however some jobs might have unpredictable running times

A more conservative approach might be better than trusting historical data in such settings

Future work...

More formal analysis

Something is hard - use approximation algorithms instead of heuristics

use asymptotic PTAS

Future work...

Classify actual real world diversity in workloads

In what workloads can we expect this strategy to work?

How much does the resource requirements vary?

Future work...

Balancing fairness and efficiency

Can the system tell you how expensive fairness is?

Golden grail

For me it’s more formal analysis