Online Load Balancing with Learned Weights

Benjamin Moseley
Tepper School of Business, Carnegie Mellon University
Relational-AI

Joint work with: Silvio Lattanzi (Google), Thomas Lavastida (CMU), and Sergei Vassilvitskii (Google)
Data Center Scheduling

- Client Server Scheduling
  - Processed in $m$ machines in the restricted assignment setting (more generally unrelated machines)
- Jobs arrive over time in the online-list model
- Assign jobs to the machines to minimize makespan
Load Balancing under Restricted Assignment

- $m$ machines
- $n$ jobs
  - Online list: a job must be immediately assigned before the next job arrives
  - $N(j)$: feasible machines for job $j$
  - $p(j)$: size of job $j$ (complexity essentially the same if *unit sized*)
- Minimize the maximum load
  - Optimal load is $T$
Online Competitive Analysis Model

- $c$-competitive
  \[
  \frac{ALG(I)}{OPT(I)} \leq c
  \]
- Worst case relative performance on each input $I$

- Problem well understood:
  - A $\Omega(\log m)$ lower bound on any online algorithm
  - Greedy is a $O(\log m)$ competitive algorithm [Azar, Naor, and Rom 1995]
Beyond Worst Case

• Reasonable assumption:
  • Access to job traces

• Desire a model to assist in assigning future jobs based on the past.
  • Predict the future based on the past.
  • What should be predicted?
  • How can it be predicted?
Learning and Online Algorithms

- Combining learning and optimization
  - Caching [Lykouris and Vassilvitskii 2018]
  - Ski Rental [Purohit et al 2018]
  - Non-clairvoyant scheduling [Purohit et al 2018]
Building a Model

- Guiding principals
  - **Computable** based on prior job traces
  - Predictions should be reasonably sized
  - Should be robust to error or inconsequential changes to the input
  - Focus on quantity to predict
    - Independent of learning algorithm used to construct the prediction
    - Focus on the worst case with access to the prediction
  - Goal: beat $\log(m)$ when error is small
    - Competitive ratio should depend on the error
What to Predict?

- **Load** of the machines in the optimal solution?
- Perhaps we can identify the contentious machines?
What to Predict?

• **Load** of the machines in the optimal solution?

• Perhaps we can identify the contentious machines? **No**

![Bar chart showing the load of machines in the optimal solution.](chart)

- New instance padded with dummy jobs loads the **same**.
What to Predict?

- **Number** of jobs that can be assigned to a machine?
  
  - Perhaps machines that can be assigned more jobs are more contentious?
What to Predict?

- **Number** of jobs that can be assigned to a machine

- Consider the following gadget to any instance

  New jobs can be assigned to old machines, skewing ‘degrees’ adversarially

New jobs say have a private machine.
What to Predict?

• Distribution on job types

• Is this the best predictive model?
  • $2^m$ job types possible
  • Need to predict a lot of information in some cases
  • Perhaps not the right model if information is sparse
What to Predict?

• Predict dual variables

• Known to be useful for matching in the random order model [Devanur and Hayes, Vee et al.]

  • Read a portion of the input
  
  • Compute the duals
  
  • Prove a primal assignment can be (approximately) constructed from the duals online
  
  • Use duals to make assignments on remaining input
What to Predict?

• Predict **dual variables** for makespan scheduling
  
  • Can derive primal based on dual
  
  • Sensitive to small error (e.g. changing a variable by a factor of $1/n^{1/2}$ has the potential to drastically change the schedule)
What to Predict?

• Idea: Capture **contentiousness** of a machine
  
  • Seems like the most important quantity besides types of jobs
Machine Weights

• Predict a weight for each machine
  • **Single number** (compact)
  • Lower weight means more restrictive machine
  • Higher weight less restrictive

• Framework:
  • Predict machine **weights**
  • Using to construct **fractional** assignments
  • **Round** to an **integral** solution online
Results on Predictions

• Existence of weights

• Theorem 1: Let $T$ be optimal max load. For any $\varepsilon > 0$, there exists machine weights and a rule to convert the weights to fractional assignments such that the resulting fractional max load is at most $(1+\varepsilon)T$.

• Theorem 2: Given predictions of the machine weights with maximum relative error $\eta > 1$, there exists an online algorithm yielding fractional assignments for which the fractional max load is bounded by $O(T \min\{\log(\eta), \log(m)\})$. 
Results on Rounding

- **Theorem 3**: There exists an **online** algorithm that takes as **input fractional assignments** and **outputs integer assignments** for which the maximum load is bounded by $O((\log \log(m))^3 T')$, where $T'$ is maximum fractional load of the input. The algorithm is randomized and succeeds with probability at least $1 - 1/m^c$.

- **Corollary**: There exists an $O(\min\{(\log \log(m))^3 \log(\eta), \log m\})$ competitive algorithm for restricted assignment in the online algorithms with learning setting.

- **Theorem 4**: Any **randomized** online rounding algorithm has worst case load at least $\Omega(T' \log \log m)$.
Existence of Good Weights

• Each machine $i$ has a weight $w_i$

• Job $j$ is assigned to machine $i$ fractionally as follows:

$$x_{i,j} = \frac{w_i}{\sum_{i' \in N(j)} w_{i'}}$$
Existence of Good Weights

- There exists weights that satisfy the following for all machines $i$
  \[
  \sum_{j} x_{i,j} \leq (1 + \epsilon)T
  \]

- Existence builds from [Agrawal, Zadimoghaddam, Mirrokni 2018]

- Used for approximate maximum matching
Finding the Weights

• Algorithm sketch for computing weights \textit{given an instance}
  
• Initialize all weights to be the same

• While there is an overloaded machine

• For each machine \( i \)
  
• Current load of machine \( i \): \[ L_i = \sum_j x_{i,j} = \sum_j \frac{w_i}{\sum_{i' \in N(j)} w_{i'}} \]

• If \( L_i \geq (1 + \epsilon)T \)
  
• Divide \( w_i \) by \( (1 + \epsilon) \)
Accounting for Error in the Predicted Weight

• Say we are given a prediction $\hat{w}$

• Let the error be the maximum $\eta = \max_i \frac{\hat{w}_i}{w_i}$

• If a machine is overloaded, run an iteration of the weight computation algorithm online
  • Converges in $\log \eta$ steps
  • If the load is greater than a $\log m$ factor off then revert to another online algorithm (i.e. greedy)

• Get a fractional makespan at most $O(T \min\{\log \eta, \log m\})$
Setup for Rounding Algorithm

- Jobs arrive online
- When $j$ arrives it reveals all $x_{i,j}$ over all machines $i$
- Assign each job immediately when it arrives
- Compare maximum load to the maximum factional load seen so far
Rounding Algorithm

• Possible approaches
  
  • Prior LP rounding techniques
    
    • Techniques are too sophisticated to be used online i.e. [Lenstra, Shmoys, Tardos 1990] needs a basic solution, BFS on support graph,…

  • Deterministic rounding
    
    • We show a $\Omega(\log m)$ lower bound

  • Vanilla randomized rounding
    
    • Easy to construct instances where a machine is over loaded by $\Omega(\log m)$
Rounding Algorithm

• Use randomized rounding with deterministic assignments

• Assign jobs to machines using the distribution defined by the fractional assignment

• If a job picks a machine with load more than $Tc \log \log m$
  - $c$ is some constant
  - The job fails

  - Let $F$ be the set of failed jobs

• Assign failed jobs using greedy (i.e. assign to the the least loaded feasible machine)
Analysis of the Rounding Algorithm

• Assume jobs (machines) have at most $\log m$ machines (jobs) in the support of their fractional assignment.

  • Most interesting case

• Only care about failed jobs (others have small makespan)

• Consider conceptually creating a graph $G$

  • Nodes are failed jobs

  • Two jobs are connected if they share the same machine
Greedy on Failed Jobs

• Prove components have polylogarithmic size, say $O(\log m)$ with high probability

• Greedy is an $O(\log m')$ approximation for an instance with $m'$ machines

  • Each component is a separate instance with number machines $m' = \text{polylog } m$

• Greedy gives a $O(\log m') = O(\log \log m)$ approximation to the fractional load
Future Work

• How to combine learning with optimization

• Can predictions be used to discover improved algorithms?

• Theoretical model characterizing good predictions?

• Does there exist a generic algorithm for using data?
Thank you!

Questions?